hyväksymispäivä arvosana

arvostelija

Docker Swarmin soveltaminen reunalaskennan ohjelmistojen hallinnoinnissa

Kristian Hansson

Helsinki 28.5.2019

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Kristian Hansson

Työn nimi — Arbetets titel — Title

Docker Swarmin soveltaminen reunalaskennan ohjelmistojen hallinnoinnissa

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	28.5.2019	56 sivua + 6 liitesivua

Tiivistelmä — Referat — Abstract

Reunalaskennan tarkoituksena on siirtää tiedonkäsittelyä lähemmäs tiedon lähdettä, sillä keskitettyjen palvelinten laskentakyky ei riitä tulevaisuudessa kaiken tiedon samanaikaiseen analysointiin. Esineiden internet on yksi reunalaskennan käyttötapauksista. Reunalaskennan järjestelmät ovat melko monimutkaisia ja vaativat yhä enemmän ketterien **DevOps**-käytäntöjen soveltamista. Näiden käytäntöjen toteuttamiseen on löydettävä sopivia teknologioita.

Ensimmäiseksi tutkimuskysymykseksi asetettiin: **Millaisia teknisiä ratkaisuja reunalaskennan sovellusten toimittamiseen on sovellettu?** Tähän vastattiin tarkastelemalla teollisuuden, eli pilvipalveluntarjoajien ratkaisuja. Teknisistä ratkaisuista paljastui, että reunalaskennan sovellusten toimittamisen välineenä käytetään joko kontteja tai pakattuja hakemistoja. Reunan ja palvelimen väliseen kommunikointiin hyödynnettiin kevyitä tietoliikenneprotokollia tai VPN-yhteyttä. Kirjallisuuskatsauksessa konttiklusterit todettiin mahdolliseksi hallinnoinnin välineeksi reunalaskennassa.

Ensimmäisen tutkimuskysymyksen tuloksista johdettiin toinen tutkimuskysymys: **Voiko Docker Swarmia hyödyntää reunalaskennan sovellusten operoinnissa?** Kysymykseen vastattiin empiirisellä tapaustutkimuksella. Keskitetty reunalaskennan sovellusten toimittamisen prosessi rakennettiin **Docker Swarm** -konttiklusteriohjelmistoa, pilvipalvelimia ja **Raspberry Pi** -korttitietokoneita hyödyntäen. Toimittamisen lisäksi huomioitiin ohjelmistojen suorituksenaikainen valvonta, edellisen ohjelmistoversion palautus, klusterin laitteiden ryhmittäminen, fyysisten lisälaitteiden liittäminen ja erilaisten suoritinarkkitehtuurien mahdollisuus. Tulokset osoittivat, että Docker Swarmia voidaan hyödyntää sellaisenaan reunalaskennan ohjelmistojen hallinnointiin. Docker Swarm soveltuu toimittamiseen, valvontaan, edellisen version palauttamiseen ja ryhmittämiseen. Lisäksi sen avulla voi luoda samaa ohjelmistoa suorittavia klustereita, jotka koostuvat arkkitehtuuriltaan erilaisista suorittimista. Docker Swarm osoittautui kuitenkin sopimattomaksi reunalaitteeseen kytkettyjen lisälaitteiden ohjaamiseen.

Teollisuuden tarjoamien reunalaskennan ratkaisujen runsas määrä osoitti laajaa kiinnostusta konttien käytännön soveltamiseen. Tämän tutkimuksen perusteella erityisesti konttiklusterit osoittautuivat lupaavaksi teknologiaksi reunalaskennan sovellusten hallinnointiin. Lisänäytön saamiseksi on tarpeen tehdä laajempia empiirisiä jatkotutkimuksia samankaltaisia puitteita käyttäen.

Avainsanat — Nyckelord — Keywords
reunalaskenta, pilvilaskenta, esineiden internet, kontit, Docker, DevOps, ketterät menetelmät

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto **1**

2 Jatkuva toimittaminen ja kontit **4**

2.1 Ohjelmistoprosessit 4

2.2 Kontit 6

2.2.1 Yleiskatsaus 6

2.2.2 Docker 8

2.2.3 Konttiklusterit 10

3 Tiedon määrä ja reunalaskenta **14**

3.1 Esineiden internet, pilvilaskenta ja reunalaskenta 14

3.2 Reunalaskenta käytännössä 16

3.3 Sovellusten operointi reunalaskennassa 18

3.3.1 Motiivit, mahdollisuudet ja haasteet 18

3.3.2 Konttien harkinta reunalaskennassa 20

4 Teollisuuden ratkaisuja reunalaskennan sovellusten hallinnointiin **22**

4.1 Teollisuuden tilanne 22

4.2 Watson IoT ja Node-RED 23

4.3 Amazon Web Services IoT 24

4.4 Microsoft Azure IoT 25

4.5 Balena 26

4.6 Palveluntarjoajien ratkaisujen vertailua 27

5 Aiempaa tutkimusta konteista reunalaskennassa **29**

5.1 Konttien vaikutus sovellusten suorituskykyyn 29

5.2 Konttien toimittaminen reunalaskennassa 30

5.3 Konttiklusterit reunalaskennassa 31

5.4 Konttien optimointi ja katsauksen yhteenveto 33

6 Tapaustutkimus **35**

6.1 Tavoitteet 35

6.2 Toteutus 39

6.2.1 Tekninen yleiskuvaus 39

6.2.2 Sovelluksen tekninen toteutus ja konttikuvan koonti 40

6.2.3 Sovelluksen toimittaminen 42

6.3 Tulokset 43

7 Johtopäätökset ja pohdinta **46**

7.1 Reunalaskennan sovellusten toimittamisen ratkaisut 46

7.2 Konttiklusterien soveltuvuus reunalaskennan sovellusten operointiin . 48

7.3 Mahdollisia jatkotutkimusaiheita 50

Lähteet **52**

Liitteet

1 main.go

2 Dockerfile

3 Dockerfile.x86

4 Jenkinsfile

5 Docker-compose.yml

6 Manifest-skripti

1 Johdanto

Ohjelmistojen kehitys ja operointi yhteen nidottuna muodostavat käsitteen **DevOps** [1]. Käsite kuvaa kulttuuria, joka on kehittynyt pilvipalveluiden aikakautena. Laitteistoja ei tarvitse hallinnoida itse, vaan laskenta- ja tallennuskapasiteettia ostetaan palveluiden tarjoajilta. Kokonaisuuteen liittyy tiiviisti ketterät periaatteet, joiden perustalle DevOps on muodostunut. Monesti termi kiteytetään käytäntöön, jossa ohjelmiston kehittäjät voivat kehittämisen lisäksi olla vastuussa muutosten viemisestä tuotannossa suoritettavaan ohjelmistoon. Lisäksi ohjelmistojen valvonta ja sen mahdollistama nopea reagointi vikatilanteisiin ja ohjelmointivirheisiin on yksi DevOps-kulttuurin tavoitteista.

DevOps-kulttuuri on pääasiassa muodostunut palvelusuuntautuneiden ohjelmistojen oheen [1]. Niissä palvelimille keskitetty ohjelmisto toimii rajapintana asiakkaille. Keskitettyä ohjelmistoa voi helposti valvoa ja skaalata tarpeen vaatiessa käytön mukaan. Ohjelmistoon on vaivatonta lisätä ominaisuuksia ja korjauksia vaihtamalla suorituksessa oleva versio uuteen. Usein tämänkaltainen ohjelmistonvaihdos onnistuu virtuaalikoneiden avulla. Ohjelmisto suoritetaan siis omassa ympäristössään.

Viime vuosina virtuaalikoneiden lisäksi yhä suosituimmaksi virtualisointitavaksi on muodostunut kevyt virtualisointi, eli **kontit** (engl. *container*) [2]. Siinä missä virtuaalikoneet muodostavat kokonaisen käyttöjärjestelmän, kevyt virtualisointi muodostaa ohjelmistolle oman suoritusympäristön käyttäen kuitenkin isäntäkäyttöjärjestelmän tarjoamia ohjelmistoja taustalla. Kontit tarvitsevat vähemmän muistia ja suoritustehoa kuin virtuaalikoneet [2, 3, 4].

Suosituimpia konttiteknologioita ovat muun muassa Docker ja LXC. Konttien periaatteena on sisällyttää ohjelmisto ja sen tarvitsemat riippuvuudet yhteen siirrettävään yksikköön, jota voidaan suorittaa missä tahansa tarvittavan konttiohjelmiston omaavassa ympäristössä. Konttien tarkoituksena on yksinkertaistaa sovellusten käyttöönottoa palvelimilla. Niiden hallinnointiin on luotu päällysteverkkoja hyödyntäviä konttiklusteriohjelmistoja, jotka mahdollistavat monimutkaisten kokonaisuuksien rakentamisen useita laitteita käyttäen. Aiemmissa tutkimuksissa konttiklusterit näyttäytyvät potentiaalisina sovellusten operointiin.

Eräs mahdollisista konttien sovelluskohteista on **reunalaskenta** (engl. *edge computing*) [5], joka tarkoittaa laskennan hajauttamista pilvipalvelinkeskuksista lähemmäs tiedon lähdettä. Eräs reunalaskennan käyttötapauksista on esineiden internet (engl. *Internet of Things*, *IoT*) [6], jolla viitataan minkä tahansa esineen kytkemiseen verk-

koon. On syntynyt ajatus, että esine voidaan kytkeä internetiin sen ollessa vuorovaikutuksessa ympäristön kanssa. Esine voi esimerkiksi kerätä tietoa käytöstään tai siihen voidaan sisällyttää erilaisia toimintoja, joissa voidaan hyödyntää tietoverkkoa. Esimerkiksi älykodin valaistusta voidaan säätää etäyhteydellä tai kaupungin älykkäitä valvontakameroita voidaan hyödyntää liikenneruuhkan hallitsemisessa.

Tietokoneet reunalaskennassa ovat usein hyvin erilaisia suoritusteholtaan ja keskusmuistin määrältään pilvilaskennassa käytettyihin laitteisiin verrattuna. Usein niissä suoritettavan ohjelmiston on oltava vaatimuksiltaan kevyt. Tällaisilla alustoilla ohjelmistojen valvonnassa ja uudelleenkonfiguroinnissa tulee esiin uusia haasteita. Laitteiden kehittyessä herää kysymyksiä DevOps-kulttuurin periaatteiden soveltamisesta reunalaskennassa.

Nykyiset kuluttajamarkkinoille suunnatut korttitietokoneet (engl. *System on a Chip*, *Single Board Computer*), kuten Raspberry Pi, ovat tarpeeksi tehokkaita suorittamaan raskaampiakin ohjelmistoja. Esimerkiksi ohjelmistojen eristäminen omaan suoritusympäristöönsä voidaan saavuttaa niissä konttien avulla. Vaikka Raspberry Pi kilpailijoineen on pääasiassa tarkoitettu kuluttajamarkkinoille ja opetukseen, sen ominaisuuksia on hyödynnetty myös tutkimuksessa. Korttitietokoneiden ympärille on muodostunut merkittävä avoimen lähdekoodin kehittäjien yhteisö. Koska korttitietokoneisiin on tyypillisesti mahdollista liittää sensoreita ja aktuaattoreita ympäristön kanssa tapahtuvaa vuorovaikutusta varten, ne toimivat luonnollisina alustoina esineen liittämiselle internetiin.

Tämän tutkielman päätavoitteena on tuoda esiin konttien mahdollisuudet ohjelmistokehityksessä ja operoinnissa, sekä tarkastella niiden soveltamista reunalaskennan käyttötapauksissa. Ensimmäiseksi tutkimuskysymykseksi asetettiin: **Millaisia teknisiä ratkaisuja reunalaskennan sovellusten toimittamiseen on sovellettu?** Vastaus koostettiin tarkastelemalla teollisuuden tarjoamia ratkaisuja. Toiseksi tutkimuskysymykseksi asetettiin: **Voiko Docker Swarmia hyödyntää reunalaskennan sovellusten operoinnissa?** Kysymykseen vastattiin suorittamalla empiirinen tapaustutkimus, johon määritettiin tietyt toteutusvaatimukset, sekä tekemällä päätelmiä aiemman tutkimuksen pohjalta.

Tutkielma koostettiin kirjallisuuskatsauksesta, teollisuuden ratkaisujen tarkastelusta ja empiirisestä käytännön kokeilusta. Painotuksen ollessa jatkuvalla toimittamisella, esitetään ensin luvussa 2 ketterien menetelmien pohja, josta DevOps on osittain muodostunut. Kontteja tarkasteltaessa eritellään yksityiskohtaisemmin niiden käytäntöjä. Luvussa 3 määritellään reunalaskenta mahdollisena ratkaisuna pilvi-

laskennan riittämättömyydelle tiedonmäärän kasvaessa. Esineiden internetin käyttötapausten avulla käydään läpi reunalaskennan hyötyjä ja haasteita sovellusten toimittamisessa. Luvussa 4 käsitellään teollisuuden tilannetta esittelemällä pilvipalveluiden ratkaisuja reunalaskentaan. Tämän avulla luodaan kuva nykyaikaisista käytännön teknologioista, joita on jo sovellettu teollisuudessa. Luvussa 5 tuodaan esille aiempaa tutkimusta, joka käsittelee konttien käyttöä reunalaskennan ja esineiden internetin laitteissa. Luvussa 6 keskitytään pienimuotoiseen empiiriseen tapaustutkimukseen, jossa rakennettiin ohjelmiston operointikokonaisuus reunalaskentaan konttiklusteria, korttitietokoneita ja pilvipalvelimia käyttäen. Tapaustutkimuksen tarkoituksena on osoittaa aiemmassa tutkimuksessa esiintyvien ratkaisumallien soveltuvuus käytännössä. Luvussa 7 vastataan tutkimuskysymyksiin ja pohditaan niiden merkistystä suhteessa aiempaan tutkimukseen. Lopuksi ehdotetaan mahdollisia jatkotutkimusaiheita.

2 Jatkuva toimittaminen ja kontit

Jatkuva toimittaminen (engl. *continuous delivery*) on yksi ohjelmistoprosessin käytännöistä. Sillä tavoitellaan ohjelmistojen julkaisemista mahdollisimman nopeasti. Tässä luvussa perehdytään jatkuvaan toimittamiseen ketterien ohjelmistoprosessien näkökulmasta. Lisäksi käsitellään tarkemmin konttien mahdollistamia käytäntöjä ja hyötyjä nykyaikaisessa palvelusuuntautuneiden ohjelmistojen kehityksessä.

2.1 Ohjelmistoprosessit

Nykyaikaisessa ketterässä ohjelmistokehityksessä painottuu iteratiivisuus ja inkrementaalisuus [7]. Ohjelmiston muutoksiin pyritään reagoimaan mahdollisimman nopeasti. Samalla ylläpidetään tuntemusta alasta (engl. *domain*), jolle ohjelmistoa kehitetään. Ketterän ohjelmistokehityksen julistus (engl. *agile manifesto*)[1] ilmestyi alun perin vuonna 2001. Sen tavoitteena on asettaa asiakkaan tyytyväisyys etualalle, jolloin ohjelmiston loppukäyttäjien ja tuoteomistajan rooli toiminnan hyväksymisessä korostuu. Erilaiset ketterät ohjelmistokehitysmenetelmät, kuten **Scrum** [8] ja **XP** [9] ovat kehittyneet näille periaatteille.

Ohjelmistojen vaatimukset muuttuvat joskus kehittämisen aikana. Siksi kehitystä on suoritettava osissa lisäten tarvittavia toiminnallisuuksia muutama kerrallaan, samalla ylläpitäen tuotantoa vastaavaa versiota ohjelmistosta. Laadun takaamiseksi on suotavaa, että asiakas pääsee kokeilemaan käytännössä ohjelmistoa ja esittämään toivomuksia kehityksen aikana. Ketterien ohjelmistokehitysmenetelmien käyttöönoton on todettu lisäävän muun muassa projektin sisäistä läpinäkyvyyttä ja ohjelmiston sisäistä dokumentaatiota ja vähentäneen muita koordinointimenetelmiä suurissa ohjelmistoprojekteissa [10].

Ketterien menetelmien takia on syntynyt tarvetta käytännöille, joiden avulla voi nopeasti julkaista ohjelmistojen tuotantoympäristön ja siten vastata asiakkaan toivomiin muutoksiin. Keskeiseksi käytännöksi on muodostunut jatkuva ohjelmistokehitys, johon liittyy **jatkuva integraatio** (engl. *continuous integration*) [11], **jatkuva toimittaminen** [12] ja **jatkuva käyttöönotto** (engl. *continuous deployment*) [13]. Jatkuvan integraation ajatuksena on ohjelmiston uuden tuotantoversion julkaiseminen mahdollisimman usein. Kehittäjät saattavat lisätä muutoksia ohjelmistoon useita kertoja päivässä. Jatkuva toimittaminen taas on tuotantoversion automaat-

[1] https://agilemanifesto.org/

tista testausta, jonka onnistuessa manuaalinen julkaiseminen on sallittua. Jatkuvan käyttöönoton periaatteena on automatisoida testaamisen ja laadunvalvonnan lisäksi tuotannon päivittäminen, mikä tarkoittaa lähes koko prosessia [14].

Edellä mainitut jatkuvat käytännöt keskittyvät ohjelmistojen toimittamisen eri osioihin, kuten versionhallintaan ja lopulta käyttöönottoon. Jatkuvat ohjelmistokehityskäytännöt eivät kuitenkaan rajoitu vain toimittamiseen, vaan ohjelmiston jatkuva valvonta kuuluu nykyaikaisten palveluiden hallintaan [15]. Valvonnalla tarkoitetaan suorituksessa olevan ohjelmiston käyttäytymisen tutkimista, jonka avulla voidaan reagoida usein ja nopeasti palvelun laatuun vaikuttaviin tekijöihin.

DevOps-kulttuuri on muodostunut ketterien menetelmien ja jatkuvan ohjelmistokehityksen käytäntöjen ympärille. Sen voidaan määritellä olevan ohjelmistokehityksen ja hallinnoinnin yhteenliittämistä laadun varmistamisen, ketterän toimittamisen ja käyttöönoton takaamiseksi [16]. Nimi DevOps muodostuu englanninkielisistä sanoista *development* ja *operations*, kehitys ja operaatiot. Termi ilmentää niiden välisen kuilun kapenemista. Nykyaikaiset pilvipalveluiden tarjoajat ovat mahdollistaneet DevOpsin synnyn, sillä palvelintilan vuokraaminen on vaivatonta [16]. Varsinainen ohjelmointityö on esimerkki kehityksestä. Jatkuva integraatio, toimittaminen, käyttöönotto ja valvonta ovat puolestaan esimerkkejä operaatioista. DevOps-periaatteiden soveltaminen on todettu tehostavan ohjelmistotuotantoprosessia yrityksissä [17].

DevOpsiin sisältyvien jatkuvan integroinnin ja toimittamisen vuoksi ohjelmiston kehitysympäristö ja tuotantoympäristö pyritään pitämään samankaltaisina. Ohjelmistojen kehittäjät käyttävät erilaisia työkaluja, mikä haastaa kehittämisen sujuvuutta. Ohjelmointivirheitä tulisi olla mahdollisimman vähän valmiissa tuotteessa. Kehitysympäristöjen samankaltaistamiseta varten on kehitetty erilaisia virtualisointi- ja hallinnointityökaluja, kuten Ansible[2] ja Puppet[3].

Jatkuvaan toimittamiseen ja integrointiin on luotu useita käytäntöä helpottavia työkaluja, joiden avulla ohjelmistokehitysprosessia voidaan hallinnoida. Ensinnäkin ohjelmiston lähdekoodi säilötään versionhallintajärjestelmään. Tarkoituksena on ylläpitää rekisteriä ohjelmakoodiin tehdyistä muutoksista. Kehittäjät voivat katselmoida toistensa tekemiä muutoksia ennen niiden hyväksymistä. Eräs suosiota saavuttanut versionhallintajärjestelmä on Git[4]. Sen mahdollistama puumainen työnkulku

[2]https://www.ansible.com/
[3]https://puppet.com/
[4]https://git-scm.com/

alkaa usein kehityshaaran eriyttämisellä tuotantohaarasta. Kehityshaaraan tehdään muutoksia, jotka yhdistetään takaisin tuotantohaaraan, mahdollisesti toisen kehittäjän katselmoinnin ja hyväksynnän myötä.

Versionhallinnan lisäksi käytäntöä helpottamaan on kehitetty automaatiopalveluita, jotka ovat nimensä mukaisesti ohjelmistokehityksen automatisointiin keskittyneitä ratkaisuja. Esimerkkejä automaatiopalveluista ovat Jenkins[5] ja Travis[6]. Ne tarjoavat käyttöliittymän ja erilaisia toiminnallisuuksia projektien toimittamisen ja laadunvarmistamisen automatisointiin. Ne voidaan määrätä suorittamaan automaattisesti usein toistuvat tehtävät, kuten yksikkö- ja integraatiotestauksen, sovelluksen koonnin ja sen käyttöönottamisen tuotannossa.

Automaatiopalvelun voi tavallisesti asettaa seuraamaan versionhallintarepositorioon tulevia muutoksia ja reagoimaan niihin. Esimerkiksi Git-repositorion tuotantohaaran päivittyessä palvelu voi ladata ja kääntää ohjelmakoodin, suorittaa automaattiset testit ja toimittaa valmiin ohjelmiston tuotantoon. Ennen toimittamista automaatiopalvelu voidaan asettaa suorittamaan ohjelmiston toimintaa varmistavat automaattiset testit, joilla tarkistetaan voiko toimittamisen suorittaa loppuun. Automaatiopalvelujen käyttö ei kuitenkaan varmista ohjelmistojen laatua itsestään, vaan kehittäjien, testaamiseen erikoistuneiden henkilöiden, tuoteomistajien ja ylläpitäjien on määriteltävä ohjelmiston hyväksymiskriteerit.

2.2 Kontit

Jatkuvassa toimittamisessa ohjelmiston suoritusympäristöjen erilaisuus näyttäytyy haasteena. Tässä aliluvussa tarkastellaan kontteja vaihtoehtona virtuaalikoneille. Konttien hyödyt sovellusten toimittamisen välineenä tuodaan esille. Lopuksi käydään läpi konttien käyttöä konkreettisilla esimerkeillä ja esitellään konttiklusterit.

2.2.1 Yleiskatsaus

Kehittäjätiimillä voi olla käytössään useita erilaisia käyttöjärjestelmiä. Ohjelmistojen suoritusympäristöjen poikkeavuus toisistaan on keskeinen ongelma suurissa ohjelmistokehitysprojekteissa. Ohjelmistot sisältävät usein kolmannen osapuolen kehittämiä riippuvuuksia, kuten ohjelmointikirjastoja, joiden asentaminen on varmistettava sekä kehityksen että tuotannon aikana. Sovelluksen kehityksessä saatetaan

[5]https://jenkins.io/

[6]https://travis-ci.org/

käyttää tiettyä versiota ohjelmointikielestä, joka eroaa muista versioista. Lisäksi ohjelmiston toimintaan kuuluu oleellisesti kommunikaatio muiden ohjelmistojen kanssa. Esimerkiksi web-palveluiden sovellukset käyttävät tietokannanhallintajärjestelmiä.

Virtualisointi on ollut pitkään eräs menetelmistä samankaltaistaa sovellusten suoritusympäristöä. Perinteisesti virtualisointia toteutetaan niin kutsutuilla virtuaalikoneilla, jotka luovat kuvan kokonaisesta käyttöjärjestelmästä, virtualisoiden koko laitteiston. Virtuaalikoneiden hallintatyökaluja (engl. *hypervisor* tai *virtual machine monitor*) ovat muiden muassa VMWare[7] ja VirtualBox[8]. Useat pilvipalvelut hyödyntävät laitteiston virtualisointia toiminnassaan tarjoten virtuaalisia palvelimia asiakkaidensa laskennan alustaksi. Virtuaalikoneita voi hyödyntää ohjelmistokehityksen aikana suorittaen paikallisella kehityskoneella samankaltaista virtuaalikonetta kuin tuotannossa.

Viime vuosien aikana sovellusten kontittaminen (engl. *container*, *containerization*), eli käyttöjärjestelmätason virtualisointi on saavuttanut suosiota, koska kontit ovat osoittautuneet kevyemmäksi ja nopeammaksi vaihtoehdoksi virtuaalikoneille [2, 3, 4]. Perinteiset virtuaalikoneet virtualisoivat isäntäkäyttöjärjestelmän laitteiston sisällyttäen itseensä kokonaisen käyttöjärjestelmän työkaluineen ja ohjelmistoineen. Kontit suoritetaan isäntäkäyttöjärjestelmän käyttäjätilassa. Ne eivät pyri virtualisoimaan laitteistoa, vaan ainoastaan eriyttämään toimintansa isäntäkäyttöjärjestelmästä omaksi prosessikseen. Samalla ne hyödyntävät isäntäkäyttöjärjestelmän ydintä (engl. *kernel*), joka sisältää tietokoneen laitteiston käyttämiseen tarvittavat ajurit.

Konttien periaatteena on sisällyttää kaikki ohjelmistoa suorittavat riippuvuudet käyttöjärjestelmäkohtaisia alirutiineja myöten yhteen ympäristöön. Kontin on tarkoitus sisältää kaikki sovelluksen suorittamiseen tarvittavat asiat. Laite, jossa kontti suoritetaan, tarvitsee vain konttien käynnistämiseen tarkoitetun ohjelman ilman muita tuotanto-ohjelmiston sisältämiä riippuvuuksia.

Monet pilvipalveluntarjoajat hyödyntävät kontteja tarjoamissaan ratkaisuissa. Kontit mahdollistavat kehitysympäristön ja tuotantoympäristön samankaltaistamisen, joten niiden avulla voidaan ehkäistä erilaisista suoritusympäristöistä johtuvia ongelmia. Kehittäjä voi käynnistää kehitettävän palvelun samankaltaisessa kontissa kuin missä se suoritetaan tuotantoympäristössä, testaten sovelluksen toimintaa jo

[7] https://www.vmware.com
[8] https://www.virtualbox.org/

```
FROM python:3
WORKDIR /application
COPY ./src /application/src
COPY ./requirements.txt /application/
RUN pip install -r requirements.txt
CMD ["python", "src/app.py"]
```

Listaus 1: Yksinkertainen Docker-kuvan määritelmä

kehittämisen aikana. Konttien hyödyntäminen pilvipalveluissa on todettu lisäävän operoinnin tehokkuutta mikropalveluiden yhteydessä [18].

2.2.2 Docker

Yksi suosituimmista työkaluista konttien luomiseen on Docker[9]. Se on avoimen lähdekoodin ohjelmisto, jota ylläpitää samanniminen yritys. Ohjelmistosta on saatavilla maksullisia enterprise-ratkaisuja ilmaisen **Docker Community Editionin** lisäksi. Monet muut konttien hallinnointia helpottavat ohjelmistot perustuvat Docker-kontteihin. Koska Docker on suhteellisen helposti asennettavissa Raspberry Pi -korttitietokoneeseen, tämän tutkimuksen empiirisessä osuudessa (luku 6) käytettiin Dockeria. Täten Dockerin toimintoja on syytä kuvailla tarkemmin.

Docker-ohjelmistoa käytetään ensin luomalla **kuva** suoritusympäristöstä ja ohjelmistosta. Kuvasta luodaan **kontti**, joka suorittaa sille määritellyn prosessin. Tämä prosessi voi olla esimerkiksi web-palvelu tai yksittäinen, kerran suoritettava tehtävä (engl. *web service*, *batch*). Kuvasta voidaan luoda useita säiliöitä samanaikaiseen suoritukseen erilaisin keinoin. Docker-kontteja suoritetaan niin kauan kuin sille määriteltyä ohjelmistoa suoritetaan. Suorituksen jälkeen kontin voi joko poistaa tai käynnistää uudelleen.

Dockerin työnkulku on seuraavanlainen: ohjelmistolle määritellään tiedosto *"Dockerfile"*, joka sisältää määritelmän ohjelmiston pohjakuvasta, sekä tarvittavat käskyt ohjelmiston riippuvuuksien asentamiseen, itse ohjelmakoodin tai binääritiedoston lisäämiseen ja lopulta käynnistämiseen. Listaus 1 on yksinkertainen Python-projektin Docker-tiedosto. Ensimmäinen komento *FROM* määrittelee pohjakuvaksi *python:3*-kuvan, joka sisältää Python-projektin tarvitsemat työkalut, eli Python-kielen tulkin ja riippuvuuksienhallintaohjelmiston *Pip*. Työhakemistoksi määritellään säiliön juuressa sijaitseva *application*, johon kopioidaan isäntähakemistossa sijaitseva lähdekoodi, eli *src*-hakemisto. Koska Python-projektit tyypillisesti sisältävät

[9]https://www.docker.com/

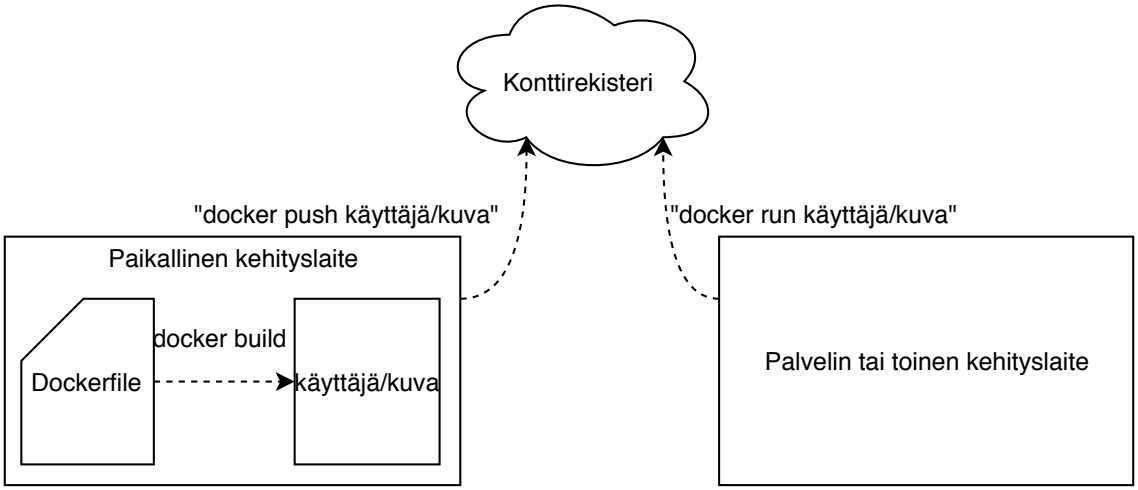
Pip-ohjelmistolla hallinnoitavia riippuvuuksia, lisätään myös riippuuvuudet listaava tiedosto *requirements.txt* konttiin. Riippuvuudet asennetaan Docker-kuvan tiedostojärjestelmään *RUN*-komentoa käyttäen. RUN-komennolle määritellään vain tavallinen Unix-komentorivikomento. Lopuksi *CMD*-komento määrittelee käynnistettävän ohjelman parametreineen. Listauksen 1 määritelmässä Python-tulkki käynnistetään parametrinaan src-hakemistossa sijaitseva *app.py*-tiedosto. Itse kuva luodaan komennolla *docker build*. Docker-asiakas voi käynnistää säiliön kyseisestä kuvasta komennolla *docker run*.

Kuvan kokoamisen jälkeen itse kuva on käytettävissä vain paikallisesti sillä laitteella, jolla kuva luotiin. Kuva koostetaan usein joko kehittäjän laitteella tai jatkuvan integroinnin palvelussa, jonka jälkeen se on siirrettävä vielä tuotantopalvelinten tai muiden kehittäjien saataville. Docker-kuvat kuuluu siirtää rekisteriin (engl. *registry*), josta asiakkaat voivat ladata ne käyttöönsä. Docker-yritys tarjoaa osittain ilmaisen rekisteripalvelun, Docker Hubin[10], joka sisältää palvelun käyttäjien luomien pohjakuvien lisäksi Dockerin tarjoamat viralliset pohjakuvat. Docker Hubissa rekisteri on jaettu käyttäjänimiin ja näiden sisältämiin repositorioihin. Esimerkiksi käyttäjän *user* kuva *api* sijaitsee rekisterissä polussa *user/api*. Kuviin on mahdollista liittää myös tunnisteita (engl. *tag*), jotka esitetään repositorion polun jälkeen kaksoispisteellä eroteltuna, esimerkiksi *user/api:v3*. Jos tunnisteen jättää määrittelemättä, Docker-ohjelmisto käyttää automaattisesti tunnistetta *latest*. Dockerin virallisissa kuvissa ei ole käyttäjänimeä. Listauksen 1 tiedostossa pohjakuvana on virallinen Python-tulkin sisältämä kuva.

Docker-rekistereitä voi rakentaa itse ja pilvipalveluntarjoajat ylläpitävät yksityisiä rekistereitä. Rekisterien lisäksi kuvia voi tallentaa fyysisesti kiintolevylle tiedostoina. Docker Hubissa on toiminto, jolla voidaan liittää jaettu versionhallintarepositorio, kuten Github[11] suoraan automaattiseen kuvan koontiin. Tällöin kehittäjän riittää vain julkaista uusi versio ohjelmiston lähdekoodista saadakseen kontin kuvan rekisteriin. Kuvassa 1 havainnollistetaan mahdollista Docker-kuvan koonnin ja kontin käynnistämisen työnkulkua kehittäjän paikalliselta kehityslaitteelta palvelimelle. Topologian keskiössä on Docker-rekisteri. Käyttäjä luo Dockerfilessa määritellyn kuvan (*build*) ja lataa sen rekisteriin (*push*). Kuvan alaosassa sijaitseva palvelin lataa kyseisen kuvan ja käynnistää siitä kontin (*run*).

Kontit ovat oletuksena tilattomia, joten ne eivät tallenna tuottamaansa tietoa muu-

[10]https://hub.docker.com/
[11]https://github.com/



Kuva 1: Dockerin mahdollinen työnkulku

alle kuin itseensä. Tällöin esimerkiksi tietokannanhallintajärjestelmää suoritettaessa Docker-konttiin tallennettu tieto häviää aina kun kontti poistetaan. Docker-konttien tilan voi kuitenkin tallentaa joko kuvaamalla osan isäntäkäyttöjärjestelmän tiedostojärjestelmästä kontin käyttöön tai luomalla oman tilan, eli volyymin (engl. *volume*) kontin käyttöön. Docker-ohjelmistossa on volyymien hallinnointiin oma toiminto. Volyymin etuna on riippumattomuus isäntäkäyttöjärjestelmän tiedostojärjestelmän rakenteesta, jolloin se on kokonaan Dockerin hallinnoima. Tällöin sitä voi helpommin käsitellä. Esimerkiksi varmuuskopioiden luominen ja volyymien siirtäminen uusiin ympäristöihin onnistuu helposti. Isäntäkäyttöjärjestelmän lisälaitteita voi kuvata (engl. *mapping, device mapping*) kontin käytettäväksi. Kontti voi hyödyntää tällöin USB-lisälaitteita tai laitteistoon sisäänrakennettua Bluetooth-rajapintaa.

2.2.3 Konttiklusterit

Suurissa palveluissa on usein tarvetta skaalata palveluiden saatavuutta horisontaalisesti, eli luoda suoritettavasta ohjelmistosta useampia ilmentymiä ja jakaa laskenta niiden kesken. Palvelukokonaisuudet voivat koostua useasta pienestä yksittäiseen tehtävään rakennetusta palvelusta, joita tulee pystyä hallinnoimaan keskitetysti. Esimerkiksi web-palveluita varten tarvitaan usein tietokannan hallintaohjelmisto, asiakkaan käyttöliittymään tarvittavat staattiset tiedostot tarjoava levypalvelin ja varsinainen ohjelmointirajapinta keskustelemaan käyttöliittymän ja tietokannan välille. Tässä tapauksessa ohjelmointirajapinta on horisontaalisesti skaalattavissa.

Oletuksena kontit eivät pysty suoraan keskenäiseen vuorovaikutukseen. Tällöim esi-

```
version: "3"
services:
  web:
    image: repositorio/python-palvelin:latest
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - webnet
networks:
  webnet:
```

Listaus 2: Esimerkki docker-compose.yml-tiedostosta

merkiksi Docker-kontteja käynnistettäessä on määriteltävä yhteys muiden konttien välille. Jokaisen kontin käynnistäminen ja hallinnointi erikseen ei kuitenkaan ole kätevää, joten näitä toimenpiteitä varten on luotu erilaisia työkaluja. Eräs tärkeimmistä Dockerin lisätyökaluista on hajautettuun laskentaan tarkoitettu Docker Swarm. Sen avulla voidaan liittää useita laitteita samaan verkkoon, eli klusteriin. Docker Swarmin avulla voidaan hallinnoida keskitetysti laitteilla suoritettavia palveluita. Sen avulla on mahdollista jakaa laskennan kuormitusta laitteiden välillä luomalla useita kontteja samaa ohjelmistoa suorittamiseen, jolloin horisontaalinen skaalaus toteutuu. Docker Swarm alustetaan ensin *swarm init* -komennon avulla halutulla hallinnoivalla laitteella (engl. *manager*). Tämän jälkeen muut laitteet liitetään hallinnoivan laitteen määrittelemään klusteriin *swarm join* -komennolla joko **tekijänä** (engl. *worker*) tai uutena hallinnoivana laitteena. Sekä hallinnoivat että tekijälaitteet voivat suorittaa samoja palveluita, mutta hallinnoivat laitteet jakavat lisäksi tehtäviä tekijälaitteille. Tehtävä voi olla esimerkiksi rajapintapyyntö. Docker Swarm sisältää valmiiksi tehtävienjakoon tarvittavat rutiinit.

Docker Swarmin avulla hallinnoidaan **palveluita** (engl. *service*), jotka ovat Dockerin käsitteistössä hajautetun laskennan yksiköitä. Ne määritellään YAML-muotoiseen tiedostoon, johon listataan jokainen tarvittava komponentti erikseen erilaisin asetuksin. Jokaiselle palvelulle määritellään joukko asetuksia, joilla sovellusta halutaan suorittaa. Listaus 2 on esimerkki yml-tiedostosta, jossa on määritettynä yksi palvelu, ”web”. Kyseisestä palvelusta luodaan viisi samaan aikaan suoritettavaa konttia (engl.

replicas), sekä rajoitetaan niiden muistinkäyttö enintään 50 megatavuun ja suorittimen käyttö kymmeneen prosenttiin. Listauksen 2 tapaisia tiedostoja käytetään sekä Docker Compose -työkalun että Docker Swarm -klusterin kanssa *stack*-komennolla. Docker Compose[12] on erillinen komentorivityökalu, jonka päätarkoitus on tehdä Dockerin käytöstä helpommin hallittavaa. Stack-komento taas on tarkoitettu helpottamaan usean samanaikaisen toisistaan riippuvaisen palvelun hallintaa Docker Swarm -klusterissa. Docker Swarmin palveluita voidaan käynnistää ja päivittää hienojakoisesti *docker service* -toiminnolla käyttäen asetusten määrityksissä erilaisia komentoriviasetuksia. Stack-komennolla voidaan hyödyntää YAML-tiedoston deklaratiivisuutta, mikä tarjoaa käyttäjäystävällisemmän kokemuksen verrattuna parametrien syöttämiseen komentorivikomennoilla. Lisäksi asetusten määritteleminen yhdessä tiedostossa voi helpottaa ohjelmistoprojektin ylläpidettävyyttä.

Docker Swarm muodostaa yhteyden laitteiden välille päällysteverkkoteknologiaa hyödyntäen, mikä mahdollistaa kommunikaation eri Docker-taustaprosessien välillä. Päällysteverkkoja on mahdollista käynnistää samaan aikaan useita eri palveluille. Niille on myös mahdollista määritellä erilaisia asetuskia. Esimerkiksi verkon tietoliikenteen voi salata tai sen kattamaa IP-avaruutta voi rajata. Palveluita voi myös jakaa eri verkkoihin jos järjestelmän osia halutaan eristää toisistaan. Verkkoja luodaan, muokataan ja poistetaan *network*-työkalulla Dockerin komentoriviohjelmaa käyttäen. Verkkojen asetuksia on mahdollista määritellä deklaratiivisesti composetiedostoissa Docker Swarmin palveluiden lisäksi. Listauksen 2 tiedostossa on määriteltynä verkko nimeltään ”webnet”. Tiedostossa määritelty palvelu on sijoitettu tähän verkkoon.

Docker Swarmin lisäksi konttiklustereiden hallintaan on kehitetty muun muassa Kubernetes[13] ja Mesos Marathon[14]. Kubernetes on päällisin puolin hyvin samanlainen kuin Docker Swarm. Sen hallinnoivaa laitetta kutsutaan nimellä *master* ja klusteriin liitettyjä tekijälaitteita nimellä *node*. Kubernetes ei itsessään ole konttiteknologia, vaan se on tarkoitettu esimerkiksi Docker- tai rkt-konttien hallinnointiin. Kubernetes lisää yhden abstraktiotason konttien ympärille niiden hallinnoinnissa. Docker Swarmissa yksittäinen palvelu on käytännössä yksi kontti, jolle on määritelty käynnistämisen asetukset. Kubernetesin käsitteistössä palvelua vastaa **palko** (engl. *pod*), jossa suoritetaan yhtä tai useampaa konttia. Kolmikosta Mesos Marathon puolestaan on pääosin Apache Mesos -klusterin hallinnointiin luotu ohjelmisto. Käyttäjän

[12] https://docs.docker.com/compose/
[13] https://kubernetes.io/
[14] https://mesosphere.github.io/marathon/

näkökulmasta helpoimmin lähestyttävä klusteriteknologia on Docker Swarm, sillä se on liitettynä Docker Community Editionin asennuspakettiin, eikä muita asennuksia tarvitse tehdä.

3 Tiedon määrä ja reunalaskenta

Nykyaikaisen pilvilaskennan myötä palveluiden kehitys on tullut ketterämmäksi. Palvelut tuottavat paljon tietoa, jota analysoidaan keskitetysti. Tässä luvussa tarkastellaan tiedon lisääntymistä tulevaisuudessa ja tuodaan esiin reunalaskennan käsite käytännön esimerkein. Koska nykyaikaisia ohjelmistoja pyritään hallinnoimaan sekä kehityksen että tuotannon suorituksen aikana, keskeiseksi pohdinta-aiheeksi otetaan reunalaskennan ohjelmistojen jatkuva toimittaminen ja valvonta motivoivan esimerkin avulla.

3.1 Esineiden internet, pilvilaskenta ja reunalaskenta

Internet on ollut arkipäivää jo vuosia. Mobiiliverkkojen ja -laitteiden kehittyessä erilaiset palvelut ja sovellukset ovat tulleet yhä useamman ihmisen saataville. Älylaitteiden lukumäärä on kasvanut ja niiden laskenta- ja tallennuskapasiteetti mahdollistavat yhä monimutkaisempien sovellusten suorittamisen. Laitteiden kehityttyä yhä pienikokoisemmiksi on herännyt ajatus esineiden muodostamasta verkosta. Esine voi olla esimerkiksi huonekalu, ranneke tai ajoneuvo. Kärjistetysti voidaan sanoa, että mihin tahansa esineeseen on mahdollista lisätä tietokone. Yhä useamman esineen muodostama verkko on herättänyt ajatuksen *ohjelmoitavasta maailmasta* [19], jossa mitkä tahansa perinteisesti ei-älylliset esineet kommunikoivat keskenään lähettäen ja vastaanottaen dataa, sekä hyödyntäen sitä omassa vuorovaikutuksessaan ja toiminnassaan ympäristönsä kanssa.

Käyttötapauksia esineiden internetistä ovat muun muassa tehtaiden älykkäät tuotantokoneistot, joiden keräämää tietoa voidaan käyttää toiminnan optimointiin [20]; älykotisovellukset [21], joiden avulla voidaan säätää kodin sähkövalaistusta päivänvalon mukaan älylaitteeseen asennettua sovellusta käyttäen; puettavat laitteet, kuten älyrannekkeet [22] ja älykkäät kaupungit useine käyttötapauksineen [23]. Suuri tiedon määrä luo mahdollisuuksia erilaisille sovelluksille, kuten maantieteellisen tiedon visualisoinnlle tai tilastolliselle analysoinnille. Esineiden internet ilmenee siis tietoliikenneyhteyksien monipuolisena hyödyntämisenä erilaisissa fyysistä maailmaa koskevissa asioissa.

Perinteisesti internetiä ovat käyttäneet ihmiset, hyödyntäen palveluita ja keskustellen keskenään palvelinten välityksellä. Palvelin ja asiakas -järjestelmässä asiakas lataa palvelimelle suoraan tuottamansa tiedon, joka tallennetaan prosessointia ja mahdollista jatkokäsittelyä varten. Käytännönläheisempiä esimerkkejä ovat web-

palvelut. Niissä tieto kerätään usein ihmisiltä, eli asiakkailta, erilaisten sovellusten avulla. Esineiden internetin myötä ihmisten lisäksi tiedontuottajina toimivat esineisiin liitetyt laitteet. Esineillä on oltava mahdollisuus olla tunnistettavissa ja löydettävissä kommunikaatioprotokollan avulla. Lisäksi niiden on kyettävä jonkinlaiseen laskentaan ja mahdollisesti vuorovaikutukseen ympäristön kanssa [24].

Verkkoon liitetään yhä useampia ympäristöä havainnoivia ja mittaavia laitteita. Niiden tuottaman tiedon määrä on paljon suurempi kuin ihmisten tuottaman tiedon määrä. Vaikka yhteysnopeudet ovat kehittyneet, ne eivät ole riittäviä kaiken tiedon siirtämiseen. Globaalin palvelinkeskusliikenteen on ennustettu olevan noin kahdenkymmenen tsettatavun luokkaa vuonna 2021 [25]. Ratkaisuksi on esitetty mahdollisuutta siirtää laskentaa lähemmäs tiedon lähdettä, eli reunaa. Tähän periaatteeseen liittyvät termit sumu- [26] ja reunalaskenta [5] (engl. *edge computing*, *fog computing*). Pelkkä pilvilaskenta ei ole tulevaisuudessa riittävä kaikkiin käyttötapauksiin. Sumu- ja reunalaskennan rooli esineiden internetin yhteydessä liittyy esineiden ohjelmistojen ja niiden tuottaman tiedon hallinnointiin. Reunalaskenta on voimakkaasti kehittyvä nuori ala, johon liittyviä tutkimusmahdollisuuksia on esitetty runsaasti [27].

Reuna- ja sumulaskennalle ei ole virallista määritelmää. Termit voivat joko tarkoittaa samaa asiaa tai sumulaskenta voidaan sijoittaa pilvilaskennan ja reunalaskennan väliin. Joka tapauksessa sekä sumu- että reunalaskenta käsittelevät laskennan siirtämistä pilvipalveluiden palvelinkeskuksista lähemmäs päätelaitteita. On esitetty, että sumulaskenta käsittelee laajemmin järjestelmien infrastruktuuria ja reunalaskenta keskittyy enemmän esineisiin ja niihin liittyvään laskentaan [5]. Tässä tutkielmassa käytetään yksinkertaisuuden vuoksi ensisijaisesti termiä reunalaskenta. Aiemman tutkimuksen yhteydessä luvussa 5 esitetään ratkaisuja, joissa sumukerros ja reunakerros ovat eriteltyinä. Vaikka tämä tutkielma käsittelee ensisijaisesti esineiden internetiä reunalaskennan päämotivaationa, reunalaskennan sovellukset eivät rajoitu ainoastaan esineiden internetiin. Niiden soveltamista on harkittu niin lisättyyn todellisuuden kuin sisällönjakeluverkkoihin [28].

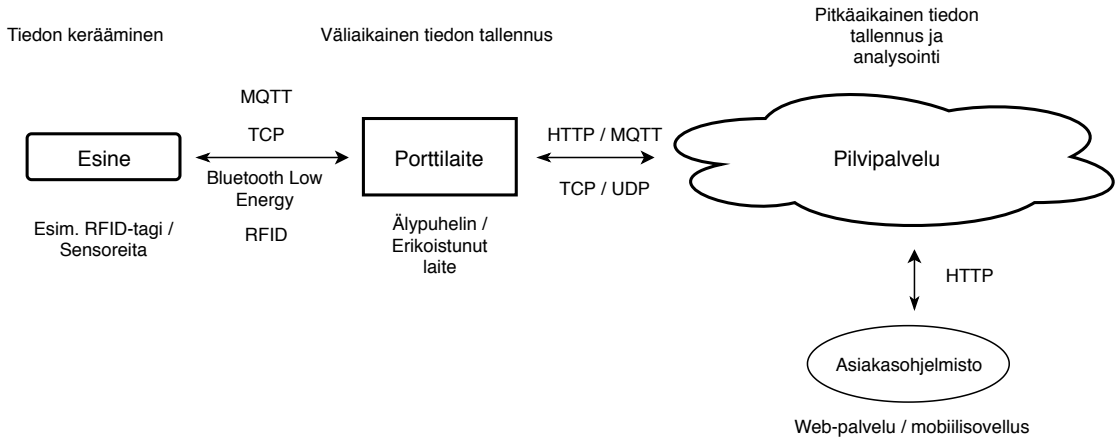
3.2 Reunalaskenta käytännössä

Esineiden internetin ratkaisujen on todettu koostuvan pääosin neljästä komponentista: **laitteista**, **yhdyskäytävistä**, **pilviohjelmistoista** ja sovelluksista (engl. *device*, *gateway*, *cloud software*, *application*) [29]. Laitteilla viitataan esimerkiksi varsinaisiin esineisiin, jotka saattavat kerätä tietoa tai vaikuttaa ympäristöönsä. Yhdyskäytävät välittävät tietoa pilviohjelmiston ja laitteiden välillä, usein tehden tarvittavaa tiedon esikäsittelyä ennen tallentamista pilveen. Pilvipalvelut toimivat tiedon pitkäaikaisen tallentamisen ja nopeaa laskentaa vaativan analysoinnin välineenä. Sovellukset taas ovat käyttäjille tarkoitettuja ohjelmistoja, jotka hyödyntävät kerättyä tietoa. Laitteiden ohjelmistoarkkitehtuurit vaihtelevat käyttötapauksen mukaan hyvin yksinkertaisista tietyn yksittäisen toiminnallisuuden toteuttavista ohjelmista kokonaisiin käyttöjärjestelmiin [29]. Tiedon siirto laitteen ja yhdyskäytävän välillä voi onnistua usealla eri tavalla. Se voi tapahtua esimerkiksi älypuhelimella kuvatun QR-koodin avulla, RFID-lukijalla tai Bluetooth-yhteydellä [30].

Reunalaite on määritelty aiemmassa tutkimuksessa tietoa käsitteleväksi laitteeksi, joka sijaitsee missä tahansa tietokeskuksen ja tiedon lähteen välissä [5]. Esineet yhdistetään usein verkkoon älykkäiden reunalaitteiden avulla sen sijaan, että ne yhdistettäisiin suoraan palvelimiin [19, 31]. Käytännössä älypuhelin voi toimia älykkäänä reunalaitteena, sillä sen avulla on mahdollista muodostaa yhteys erilaisiin laitteisiin ja samalla olla yhteydessä palveluihin [5]. Apple Watch[15] on esimerkiksi suunniteltu yhdistettäväksi iPhone-älypuhelimeen, jolloin erilaiset sovellukset voivat viestiä sen kanssa. tämän tutkimuksen yhteydessä reunalaite määritellään reunalaskennan mahdollistavaksi välineeksi, joka voi muodostaa yhteyden internetiin ja johon voi yhdistää muita laitteita, kuten sensoreita ja aktuaattoreita, kevyttä laskentaa suorittavia mikrokontrollereita tai muita reunalaitteita. Tähän määritelmään sisällytetään myös älykkäät yhdyskäytävät.

Kuva 2 esittää yksinkertaistetun esimerkin esineiden internetin ja reunalaskennan mahdollisesta arkkitehtuurista. Kuvassa luetellaan tyypillisiä teknologioita ja tietoliikenneprotokollia, joita ratkaisuissa on käytetty. Esineenä voisi olla aktiivisuusranneke, joka mittaa jatkuvasti henkilön pulssia. Rannekkeen halutaan olevan langattomassa yhteydessä käyttäjänsä älypuhelimeen. Älypuhelimeen asennettu erikoistunut sovellus tallentaa älyrannekkeen tuottamaa tietoa. Käyttäjälle halutaan tarjota mahdollisuus tarkastella automaattisesti omaa aktiivisuushistoriaansa hyvin pitkältäkin aikaväliltä. Tiedon ei kuitenkaan haluta täyttävän älypuhelimen rajat-

[15] https://www.apple.com/watch/



Kuva 2: Esimerkki esineiden internetin arkkitehtuurista

tua tallennustilaa, ottaen huomioon muut sovellukset ja käyttötarkoituksen. Lisäksi älypuhelin voi hajota, jolloin ainoastaan siihen tallennettu tieto katoaa. Älypuhelimen sovellukseen kehitetään ominaisuus, jolla käyttäjän tieto synkronoidaan periodisesti henkilökohtaisilla tunnuksilla saavutettavaan taustapalveluun. Kokonaisuuteen voidaan lisätä mobiilisovelluksesta erillinen web-palvelu, jonka avulla käyttäjän on mahdollista lukea rannekkeensa keräämää tietoa tai hyödyntää mahdollisia lisäominaisuuksia.

Aktiivisuusrannekkeen suunnittelussa on otettava huomioon laitteen akun käyttöajan kesto, joten laskennallisesti raskaimpien ohjelmistojen ja tietoliikenneprotokollien käyttöä on syytä välttää. Älypuhelimen ja rannekkeen välisessä kommunikaatiossa voidaan hyödyntää *Bluetooth Low Energy* -teknologiaa (*BLE*), joka mahdollistaa lyhyen kantaman yhteyden vähäisellä virrankulutuksella. Älypuhelimeen asennettu sovellus skannaa ympäristönsä etsien mahdollisia yhdistettäviä laitteita. Yhteys muodostetaan halutun laitteen löytyessä, jolloin sovelluksen on mahdollista pyytää mittaustietoa rannekkeelta. BLE sopisi hyvin aktiivisuusrannekkeen käyttötapaukseen. Sen lisäksi on kehitetty erilaisia kevyitä protokollia kuten MQTT ja CoAP, jotka on suunniteltu laitteiden väliseen kommunikaatioon (engl. *Machine to Machine, M2M*) energiankulutus ja mahdolliset laitteistojen rajoitukset huomioon ottaen. Tiedon siirtäminen älypuhelimesta taustapalveluun voi onnistua raskaampaa protokollaa, kuten HTTP:tä tai WebSocketia hyödyntäen. Tältä osin sovellus toimisi hyvin samalla tavalla kuin web-palveluiden asiakasohjelmistot.

Aktiivisuusrannekkeen esimerkki on vain yksi mahdollinen tapa rakentaa esineiden internetin kokonaisuus. Siihen kuitenkin sisältyy esineiden internetin palvelukoko-

naisuuden tyypilliset osat, kuten esine, yhdyskäytävä ja pilvi. Esineiden määrä ei välttämättä rajoitu vain yhteen, vaan yhdyskäytävään voi olla mahdollista yhdistää useita laitteita, kuten rakennuksessa sijaitsevia lämpösensoreita. Laitteet voivat sijoittua maantieteellisesti laajalle alueelle. Lisäksi mahdollinen asiakasohjelma voisi olla suoraan yhteydessä yhdyskäytävään niin, että tietoa luetaan ennen sen siirtämistä pilvipalvelimille. On mainittava, että esine voi itse toimia yhdyskäytävänä monissa tapauksissa olemalla suoraan yhteydessä taustapalveluihin ilman erillistä yhdyskäytävää. Tulevaisuudessa 5G-mobiiliverkon ja IPv6-protokollan yleistyessä yhä useampia laitteita voidaan yhdistää suoraan internetiin [19].

Esineiden internetin sovellukset poikkeavat tavanomaisesta sovelluskehityksestä laitteiden määrän takia. Yksittäisen mobiilisovelluksen toteuttaminen on suhteellisen yksinkertaista verrattuna esineiden internetiin, jossa suunnittelu on laajennetava sovellustasolta järjestelmätason ajatteluun [32]. Esineiden internetin ja reunalaskennan kokonaisuutta kaavailtaessa ei voi välttyä hajautetun järjestelmän rakentamiselta. Ongelmia esiintyy niin verkon heterogeenisyyden kuin ohjelmistojen uudelleenkonfiguroinnin osalta.

3.3 Sovellusten operointi reunalaskennassa

Tässä aliluvussa käydään läpi, miksi sovelluksia tulisi toimittaa ketteriä periaatteita mukaillen. Perusteluksi annetaan motivoiva esimerkki. Lisäksi eritellään reunalaskennan sovellusten hallinnoinnissa esiin nousevia haasteita. Lopuksi käsitellään kontteja toimittamisen mahdollistavana teknologiana.

3.3.1 Motiivit, mahdollisuudet ja haasteet

Reunalaskennan ja esineiden internetin projektien tarpeet ovat alati muuttuvia. Tällöin mahdollisuudet vaikuttaa reunalaskennan infrastruktuurin jatkuvaan muutokseen tulevat esille. Esimerkiksi uusia esineitä voidaan lisätä, laitteiden välinen tietoliikenneprotokolla voidaan muuttaa tai analysointimenetelmiä halutaan vaihtaa. Laitteiden kehittyessä yhä monipuolisemmiksi niiden tarkoitusta ja tehtävää voi muuttaa. Niihin voi myös lisätä yhä enemmän toimintoja ja ominaisuuksia. Kuluttajamarkkinoille kehitettävien tuotteiden laitteisto on helpompaa hankkia niin sanotusti ”suoraan kaupan hyllyltä”, sillä tiettyyn tehtävään erityisesti kehitettyjen laitteistojen tuotanto on oma kulueränsä. Saatavilla on jo valmiita edullisia laitteistokokonaisuuksia, joissa on kaikki tarvittava tuotteen kehittämiseen.

Palveluohjelmistojen kehityksessä uusien ominaisuuksien lisääminen, ohjelmointivirheiden korjaaminen ja sovelluksen kirjastoriippuvuuksien päivittäminen toimivat jo luonnollisena motiivina ohjelmistojen jatkuvaan toimittamiseen. Esineiden internetin yhteydessä motivoivana esimerkkinä voidaan kuvitella rakennuksen lämpötilaa mittaava järjestelmä, joka koostuu aluksi rakennuksen huoneisiin sijoitetuista lämpösensoreista, yhdestä tai useasta yhdyskäytävästä ja pilvipalvelussa sijaitsevasta palvelinohjelmistosta tiedon pitkäaikaista tallennusta varten. Kokonaisuuteen on luotu rakennuksen lämpötilaa visualisoiva web-sovellus, joka näyttää lämpötilan reaaliaikaisesti. Sensorit ovat yhdistettynä laitteisiin, jotka ovat yhteydessä älykkääseen yhdyskäytävään rakennuksen sisäisen lähiverkon kautta, viestittäen mittaamansa datan MQTT-protokollan avulla.

Ajan kuluessa herää ajatus, että rakennuksen ilmastointilaitteita voitaisiin hallita etäyhteydellä hyödyntäen lämpöjärjestelmän tuottamaa tietoa. Käytännöllisintä olisi päivittää olemassaolevaa järjestelmää lisäämällä toimintoja älykkäiden yhdyskäytävien ohjelmistoihin. Lisäksi keskitetty päivittäminen ja ylläpito mahdollistaisivat kaikkien järjestelmän laitteiden sovellusten päivittämisen kerralla. Ilmastointilaitteistoa ohjaava ohjelmisto on vain kyettävä yhdistämään lähiverkkoon, jossa lämpötilaa mittaavat sensorit ja älykkäät yhdyskäytävät, eli reunalaitteet sijaitsevat. Reunalaitteet voivat myös sijaita maantieteellisesti laajalla alueella, joten niiden sovellusten päivittäminen paikan päällä ei ole vaihtoehto.

Edellä esitetty esimerkki on hyvin yksinkertaistettu, mutta potentiaalinen skenaario. Tarkoituksena on huomioida, että laskentaa on mahdollista lisätä yhä useampiin käyttötarkoituksiin. Suurissa järjestelmäkokonaisuuksissa uudelleenkonfiguroinnin riski on suositeltavaa ottaa huomioon ajoissa. Laskennan siirtyminen lähemmäs tiedon lähdettä mutkistaa edelleen järjestelmän suunnittelua, sillä kehityksen kohteena ei ole pelkästään palvelinohjelmisto, vaan esineiden kanssa kommunikoivien reunalaitteiden ohjelmistot. Lisäksi, koska reunalaskennan tarkoitus on siirtää laskentakuormaa pois palvelimilta suuren tietomäärän takia, tiedon analysointi paikan päällä lähellä tiedon lähdettä voi olla tarpeellista. Reunalaite saattaa toimia myös käyttöliittymänä reaaliaikaisen tiedon näyttämiselle sisältäen esimerkiksi web-palveluohjelmiston. Tällöin voi olla tarvetta lisätä toimintoja käyttöliittymään. Lyhyesti kärjistäen, keskitettyyn sovellusten toimittamiseen reunalaskennassa on useita syitä.

Reunalaskennassa sovellusten toimittamisessa kohdataan useita haasteita. Jatkuvan toimittamisen saavuttamiseksi on toteutettava infrastuktuuri, jonka avulla ohjelmis-

tosta julkaistaan automaattisesti tuotantoversio pyrkien vähentämään mahdollisia palvelukatkoksia. Automatiikkaan kuuluu ohjelmiston laadunhallinta ja varmistus sen toimivuudesta. Ohjelmiston on siis oltava testattavissa ennen tuotantoonvientiä. Lisäksi ohjelmiston suorituksenaikaisen käyttäytymisen valvonta on tarpeellista. Valvontaa tarvitaan erityisesti, jos ohjelmistolla on useita erilaisia käyttötapauksia, esimerkiksi sensoritiedon kerääminen ja sen visualisointi reaaliaikaisesti. Eräs mahdollinen motiivi valvonnalle on sensoriyhteyden häviämisen yhteyden havaitseminen virhetilanteissa. Tiedon visualisointia voidaan käsitellä palveluna, jonka käyttöä mitataan valvonnan avulla.

Laitteiden ja verkkojen heterogeenisyys on tunnistettu haasteeksi esineiden internetissä [24]. Esineiden käyttämät suorittimet voivat olla toisistaan poikkeavia niin arkkitehtuuriltaan kuin laskentakapasiteetiltaan. Ohjelmiston kehittämisessä suositut AMD-käskyarkkitehtuuria käyttävät suorittimet voivat poiketa esineiden internetin ja reunalaskennassa käytetyistä tuotantolaitteista. Ohjelmisto on käännettävä erikseen esimerkiksi korttitietokoneissa ja älypuhelimissa suosituille ARM-suorittimille. Useat nykyaikaiset ohjelmointikielten kääntäjät tukevat erilaisia suoritinarkkitehtuureita, joten sovelluskehittäjän näkökulmasta ongelma ei ole välttämättä suuri.

Tuotannossa käytettävien laitteiden määrä voi olla runsas, mikä aiheuttaa nimeämisongelman [5]. Reunalaitteiden tulisi olla sekä yksilöitävissä että ryhmiteltävissä. Nimeämiskäytäntöjä on erilaisia, mutta reunalaskentaan ei ole kehitetty standardoitua mallia. Toisaalta nimeämiskäytäntö voi hyvin riippua käyttötapauksesta, mikä tekee siitä projektikohtaisen ongelman. Haasteena on tuotu myös esille, miten ohjelmisto toimitetaan laitteille [5]. Pilvipalveluiden erilaiset alustapalvelut ovat hyvin abstrahoituja. Usein niiden avulla luodaan vain uusi virtuaalipalvelin, joka sisältää jo valmiit työkalut ohjelmistojen käynnistämiseen. Reunalaite puolestaan on konkreettisempi. Laitteella on oltava yhteys sovelluksen toimittavalle taholle. Toimittamisen valvonta on tavoiteltavaa mahdollisten virhetilanteiden varalta. Palaten nimeämisongelmaan, joissain tapauksissa pitäisi olla mahdollista toimittaa sovelluksia yksittäisille laitteille tai laiteryhmille, esimerkiksi koekäyttöä varten.

3.3.2 Konttien harkinta reunalaskennassa

Koska kontittaminen on kätevä teknologia palvelusuuntautuneiden ohjelmistojen toimittamisessa, on luonnollista pohtia sen soveltamista reunalaskennassa. Motivaatio sovellusten toimittamiseen reunalaskennassa määriteltiin edellä. Palvelusuuntautuneista ohjelmistoista kertyneen kokemuksen myötä reunalaskennan sovellusten toi-

mittamisen välineeksi voidaan harkita kontteja. Konttien harkinta herättää kysymyksiä. Oletuksena on, että reunalaitteen suoritin voi olla arkkitehtuuriltaan erilainen verrattuna kehityslaitteeseen. Ensimmäisenä ongelmana on ratkaistava mitä toimenpiteitä vaaditaan, jotta sovellus voidaan kehittää suorittimella, jonka arkkitehtuuri poikkeaa tuotannossa käytettävän suorittimen arkkitehtuurista. Tähän liittyy sovellusten kehittäminen ja toimivuuden varmistaminen. Vaikka yksi konttien tehtävistä on ohjelmistojen suoritusympäristöjen samankaltaistaminen, suoritinten toisistaan poikkeavat käskyarkkitehtuurit on otettava lopulta huomioon. Tähän haasteeseen on jo olemassa ratkaisuja. Esimerkiksi Dockeria voi suorittaa usealla eri suoritinarkkitehtuurilla[16].

Toiseksi, mitä toimenpiteitä vaaditaan kontissa suoritettavan sovelluksen siirtämiseen reunalaitteisiin keskitetysti? Dockerin kaltainen ratkaisu on mahdollinen vaihtoehto renalaitteiden sovellusten toimittamiseen. Konttien kuvat ladataan rekisteriin, josta Docker-asiakasohjelma voi kopioida ne. Aina kun uusi versio kuvasta on saatavilla, päivitysprosessi voidaan toistaa. Uusi kuva ladataan, vanha kontti poistetaan käytöstä ja uudesta kuvasta luodaan uusi kontti. Kontteja, tarkemmin määriteltynä Dockeria käytettäessä, toimittamisen tapa on osittain siis määritelty. Kuvien lataaminen keskitetyltä rekisteripalvelimelta voi olla mahdollista. Kuitenkin tieto uudesta kuvasta olisi viestitettävä reunalaitteelle. On otettava huomioon laitteen sijainti, joka ei välttämättä ole samassa lähiverkossa kuin päivittävä palvelin. Esimerkiksi SSH-yhteys laitteeseen ei välttämättä onnistu, eikä laite välttämättä sisällä julkista rajapintaa ohjelmiston kutsumiseen. Laitteen tulisi itse muodostaa yhteys saatavilla olevaan palvelimeen ja täten mahdollistaa myös kahdensuuntainen kommunikaatio palvelimelta reunalaitteeseen.

Kolmanneksi, harkittaessa kontteja reunalaskennan sovellusten toimittamisessa, laitteistojen valinnassa on otettava huomioon konttien laskennallinen vaativuus. Kontit vaativat käyttöjärjestelmän toimiakseen, joten laitteiston tehon ja muistin kapasiteetin on oltava riittävä. Toisaalta tapauksissa, joissa sovellukset ovat suhteellisen monimutkaisia ilmenee tarve suurelle laskentateholle. Yhtenä reunalaitteen vaatimuksena voi olla laitteen pieni fyysinen koko. Nykyiset korttitietokoneet, kuten Raspberry Pi ovat kuitenkin kykeneväisiä suorittamaan laskennallisesti vaativia tehtäviä, näin ollen myös kontteja.

[16]https://blog.docker.com/2017/09/docker-official-images-now-multi-platform/

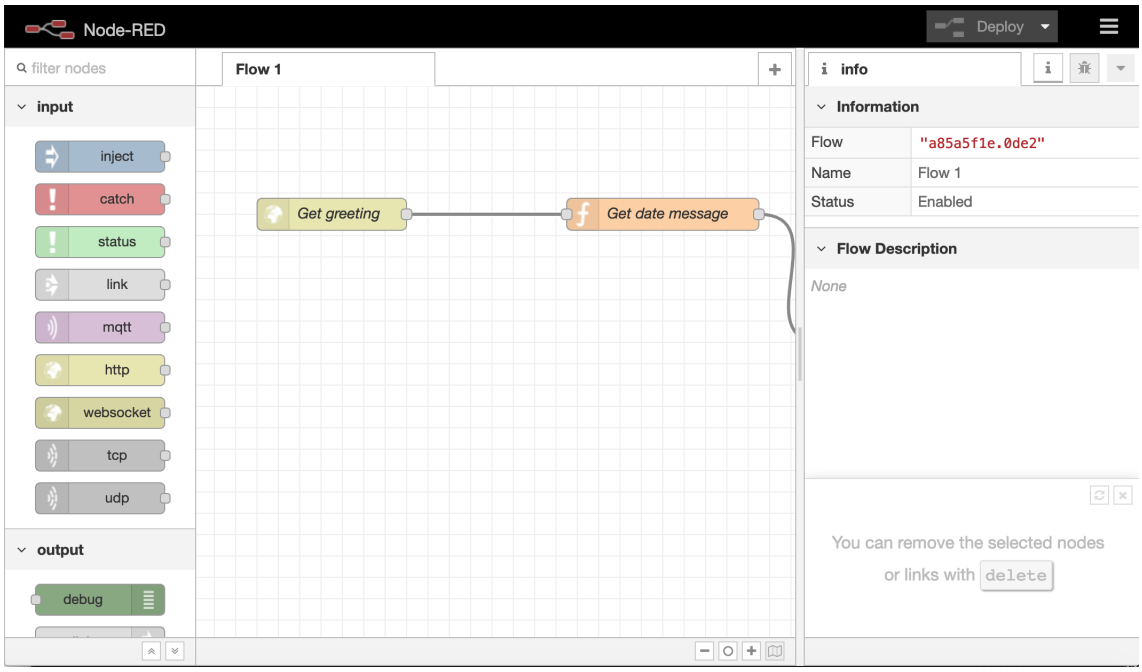
4 Teollisuuden ratkaisuja reunalaskennan sovellusten hallinnointiin

Tässä luvussa tarkastellaan olemassaolevien palveluntarjoajien mahdollistamia ratkaisuja erityisesti jatkuvan toimittamisen ja sovellusten toiminnan valvonnan näkökulmista. Tarkastelun avulla luodaan kuva yleisimmistä toiminnoista, joita reunalaskennan käyttötapauksissa otetaan huomioon. Teollisuuden ratkaisujen tarkastelu todettiin tarpeelliseksi niiden nopean kehityksen takia. Lisäksi tutkimus aiheesta on ollut toistaiseksi vähäistä.

4.1 Teollisuuden tilanne

Pilvipalveluntarjoajat ovat kehittäneet erilaisia ratkaisuja helpottamaan reunalaskennan infrastruktuurin rakentamista. Ratkaisuihin sisältyy ohjelmistoja, jotka varmistavat reunalaitteiden ja pilvipalvelinten välisen kommunikoinnin sekä valmiita alustoja laitteiden ja esineiden ohjelmointiin. Lisäksi monet palvelut tarjoavat ohjelmistoja suoritettavaksi reunalaitteissa tapahtuvaa tiedon analysointia varten. Tarkasteluun valittiin pilvipalveluntarjoajista **IBM Cloud**[17], **Amazon Web Services**[18] ja **Microsoft Azure**[19]. Koska tarkastelu tehtiin erityisesti reunalaskennan ohjelmistojen operoinnin näkökulmasta, mukaan valittiin lisäksi **Balena**[20], joka on reunalaskennan ja esineiden internetin infrastruktuuriin erikoistuneita ohjelmistoja kehittävä yritys.

Tarkastelu suoritettiin lukemalla palveluiden tarjoamaa dokumentaatiota. Näin oli mahdollista päätellä kunkin palvelun hyödyntämät teknologiat ja muodostaa kuva niiden tarjoamista mahdollisuuksista reunalaskennan sovellusten operointiin. Lopuksi palveluiden ratkaisuja vertailtiin keskenään ja tuotiin esiin yhtäläisyyksiä niiden välillä. Tarkastelu suoritettiin erityisesti sovellusten toimittamisen ja valvonnan näkökulmasta, ottaen huomioon DevOps-kulttuurin käytännöt reunalaskennassa.



Kuva 3: Node-RED-editori.

4.2 Watson IoT ja Node-RED

Watson IoT on IBM:n pilvipalvelun IBM Cloudin tarjoama esineiden internetin alusta. Pilvipalveluratkaisujen lisäksi IBM on kehittänyt reunalaitteille tarkoitetun Node.js-pohjaisen ohjelmointityökalun *Node-REDin*[21] helpottamaan esineiden ja reunalaitteiden kytkemistä toisiinsa ja pilvipalvelimiin. Sen avulla on mahdollista sekä rakentaa REST-tyylisiä rajapintoja tietokantayhteyksineen että vastaanottaa ja julkaista viestejä MQTT-protokollaa hyödyntäen. Node-RED on lisensoitu avoimen Apache 2.0 -lisenssin alle.

Node-RED asennetaan laitteeseen Node.js-ympäristön paketinhallintaohjelmistoa NPM:ää käyttäen. Node-REDin on mahdollista kommunikoida Watson IoT -alustan kanssa kahdensuuntaisesti, sekä luoda yhdyskäytäviä, joiden kautta välitetään tietoa esineistä pilvipalvelimille. Node-REDin ohjelmointi tehdään visuaalisesti selainpohjaisen graafisen editorin avulla siirtämällä ja yhdistämällä komponentteja toisiinsa. Kuva 3 esittää Node-REDin editoria käynnistämisen jälkeen.

[17] https://www.ibm.com/cloud/
[18] https://aws.amazon.com/iot/
[19] https://azure.microsoft.com/en-us/overview/iot/
[20] https://www.balena.io/
[21] https://nodered.org/

Node-RED on itsessään vain ohjelmointityökalu, jolla käsitellään reunalaitteita ja pyritään helpottamaan kokonaisuuksien hallintaa. Keskitettynä sovelluksen jatkuvan toimittamisen ratkaisuna on käytettävä muita teknologoita. Node-RED ei ole sidottu ainoastaan Watson IoT -palveluun, vaan toimii omana kokonaisuutenaan. Sitä voi käyttää muidenkin palveluiden kanssa.

IBM tarjoaa lisäksi ohjelmointikirjastoja yksinkertaistamaan yhteyden muodostamista ja tiedon siirtämistä Watson IoT -palvelun ja reunalaitteiden välillä. Kirjastoja on kehitetty sekä Node.js:lle että Pythonille. Kirjastojen lisäksi Watson IoT:lle kehitetään reunalaitteisiin suunniteltua *IoT Platform Edge* -ohjelmistoa, jonka tavoitteena on viedä pilvilaskennan toimintoja reunalaitteisiin. IoT Platform Edgen laskenta suoritetaan konteissa. Se tukee erilaisia laitteistoarkkitehtuureja, mahdollistaen sekä ARM- että AMD-pohjaisten suorittimien käytön.

Watson IoT:n tarjoamat palvelut mahdollistavat monipuolisten esineiden internetin sovellusten rakentamisen. Alusta tukee useita erilaisia laitteita ja mahdollistaa esineiden tuottaman tiedon automaattisen tallentamisen ja analysoinnin. Valmiita sovellusten toimittamisen ratkaisuja reunalaitteisiin ei ole kuitenkaan toteutettu. Käyttäjän on siis kehitettävä itse tekninen ratkaisu sovellusten toimittamiseen.

4.3 Amazon Web Services IoT

Amazon Web Services (*AWS*) tarjoaa IoT-kokonaisuudessaan reunalaitteiden ohjelmointiin *Greengrass Core* -ohjelmistoa. Greengrass Coren avulla voi toteuttaa kommunikoinnin pilvipalvelimen, reunalaitteen ja esineiden välillä, sekä suorittaa laskentaa verkon reunalla. AWS tarjoaa laskentateholtaan rajallisten laitteiden, kuten mikrokontrollerien, ohjelmointia varten *FreeRTOS*-käyttöjärjestelmää ja *IoT Device SDK* -kehityskirjastoa. Reunalaskennan ja esineiden ohjelmistokehysten lisäksi AWS:n tarjoamiin tuotteisiin kuuluu reunalaitteiden ja esineiden toiminnan hallintaa varten räätälöityjä pilvipalvelualustoja.

Greengrass asennetaan halutulle reunalaitteelle, jolle täytyy olla valmiiksi asennettuna käyttöjärjestelmä. Tuettuja alustoja ovat sekä ARM-pohjaiset korttitietokoneet että x86-käskyarkkitehtuuria käyttävät suorittimet. Ohjelmisto sisältää toiminnot, jotka yhdistävät reunalaitteen automaattisesti sille osoitettuun pilvipalvelimeen. AWS:n laitteet ryhmitellään pilvipalvelussa lähtökohtaisesti ennen Greengrassin asentamista laitteelle. Pilvipalveluun luodaan ensin ryhmä, johon halutut laitteet lisätään. Ryhmän luomisen jälkeen palvelusta on ladattava reunalaitteelle

tarkoitetut asetukset pilvipalvelimen ja reunalaitteen välistä kommunikointia varten.

Greengrassin laskenta suoritetaan yksittäisinä funktioina, jotka suorittavat tietyn tehtävän. Funktioita voidaan ohjelmoida erilaisilla ohjelmointikielillä, kuten Javalla tai Pythonilla. Toimittaminen onnistuu keskitetysti AWS:n pilvipalvelua käyttäen. Toimittamisen kohteena on käytännössä itse ohjelman ja ohjelmointikirjaston sisältämä zip-tiedosto, joka tallennetaan pilvipalvelimelle ja siirretään sen kautta reunalaitteeseen. Reunalaitteeseen voi toimittaa ohjelman lisäksi tarvittavia asetuksia tehtävien suorittamiseen. Pilvipalvelinten ja reunalaitteiden väliseen kommunikointiin käytetään MQTT-protokollaa. Funktioita voidaan suorittaa joko jatkuvina tehtävinä tai yksittäisinä suoritteina pyynnöstä. Lisäksi niissä on mahdollista hyödyntää AWS:n koneoppimispalveluita.

Toimittamisen automatisointi onnistuu AWS:n komentoriviohjelmiston avulla. Näin ollen Greengrassin sovellusten toimittamisen voi tarvittaessa lisätä osaksi kehitysprosessia, mahdollistaen näin sovellusten jatkuvan toimittamisen. Lisäksi AWS tarjoaa toiminnon, jolla on mahdollista päivittää automaattisesti Greengrassin ohjelmistot. Reunalaitteita on mahdollista ryhmittää erilaisia käyttötarkoituksia varten ja niiden toimintaa on mahdollista valvoa. AWS:n IoT-palvelu toteuttaa kattavan ohjelmistokokonaisuuden reunalaskennan mahdollistamiseksi, sisältäen toiminnot monimuotoiseen operointiin.

4.4 Microsoft Azure IoT

Microsoft Azure IoT -palvelu tarjoaa pilvilaskennan keskitettyjen palvelinratkaisujen lisäksi reunalaitteeseen asennettavan *IoT Edge runtime* -kokonaisuuden, joka sisältää reunalaskentaan tarvittavat ohjelmistot. Se koostuu pääasiassa kahdesta ohjelmistosta, *IoT Edge hubista* ja *IoT Edge agentista*. IoT Edge Hub toimii välityspalvelimena Microsoft Azuren pilvipalvelimeen tarjoten samankaltaisen rajapinnan kuin pilvessä sijaitseva *IoT hub*. Ajatuksena on, että laitteet voivat edelleen suorittaa laskentaa silloinkin kun yhteys pilvipalveluun ei ole muodostettuna. IoT Edge Hub tallentaa tiedon paikallisesti reunalaitteeseen ja synkronoi tiedon pilven ja reunan välillä yhteyden taas ollessa saatavilla. IoT Edge agent on vastuussa reunalla suoritettavien ohjelmien käynnistämisestä ja niiden tiedon välityksestä IoT hubille. Microsoft Azure IoT:n käsitteistössä varsinaisia reunalaitteissa suoritettavia ohjelmia kutsutaan moduuleiksi (*Edge module*). Edge runtime on mahdollista asentaa korttitietokoneille ja tehokkaammille laitteille. Laitteeseen on oltava asennettuna

kokonaisen käyttöjärjestelmän lisäksi Moby Engine -ohjelmisto konttien hallintaan.

Moduulit voivat suorittaa yksittäisiä funktioita tai analysoida reunalaitteen vastaanottamaa sensoritietoa. Funktioiden ohjelmointiin voi käyttää rajattua määrää tuettuja ohjelmointikieliä, kuten C#:a tai JavaScriptiä. Moduulit suoritetaan konteissa, joten niiden toimittaminen reunalaitteeseen muistuttaa konttien toimittamista palvelimelle. Microsoft Azure tarjoaa palveluissaan konttirekisterin, josta reunalaitteiden ohjelmistot lataavat moduulit sisältävät konttikuvat. Konttirekisterinä on mahdollista käyttää myös Docker Hubia. Edge Runtime tukee *Open Containers*[22] -yhteensopivia kontteja. Microsoft Azuren palvelun avulla on mahdollista ryhmittää laitteita erilaisin keinoin. Eräs mahdollisuuksista on asettaa leimoja (engl. *label*) laitteisiin. Toimittamisen keskiössä on manifestitiedosto, jossa määritellään mikä sovelluskontti toimitetaan ja mihin.

Toimittaminen on mahdollista toteuttaa automaattisesti graafisin käyttöliittymin, joten liittäminen osaksi kehitys- ja operointiprosessia on mahdollista. Microsoft Azure tarjoaa hallinnointiin valmiita ratkaisuja, jotka ovat yhteensopivia IoT-ratkaisujen kanssa. Tällainen on esimerkiksi valmiiksi asennettu Jenkins-automaatiopalvelu. Reunalaitteita on mahdollista ryhmitellä, ja yksittäiselle laitteelle on mahdollista toimittaa tietty moduuli. Sekä reunalaitteiden että niissä suoritettavien moduulien toimintaa voidaan valvoa keskitetysti. Kommunikointi reunalaitteen ja IoT Hubin välillä toimii joko MQTT- tai AMQP-protokollaa käyttäen. IoT Hubin lisäksi laitteita valvotaan IoT Central -ratkaisun avulla.

4.5 Balena

Balena[23] on konttiteknologiaan keskittynyt kokoelma työkaluja, joiden tarkoituksena on ratkaista sovellusten kehittämisen ja toimittamisen ongelmia reunalaskennassa. Ohjelmistokokonaisuutta kehittää ja ylläpitää samanniminen kansainvälinen yritys. Balenan kehittämiin ohjelmistoihin lukeutuu kevyt Linux-pohjainen käyttöjärjestelmä ja konttien hallintaohjelmisto, joka tukee Dockerin konttikuvia. Lisäksi Balena tarjoaa palvelun reunalaitteiden infrastruktuurin pystyttämiseen.

Sovellusten toimittamisessa hyödynnetään Git-versionhallintajärjestelmää ja konttirekisteriä. Toimittaminen muistuttaa hyvin paljon palvelusuuntautuneiden ohjelmistojen vastaavaa prosessia. Balena mahdollistaa sovelluksen toimittamisen suo-

[22]https://www.opencontainers.org/

[23]https://www.balena.io/

raan versionhallinnasta tuotantoon. Toisin sanoen järjestelmä toteuttaa sovelluksen automaattisen käyttöönoton monille kehittäjille tuttujen työkalujen avulla.

Balenan tarjoama palvelu koostuu pääosin keskitetystä Git-palvelimesta ja konttirekisteristä. Palveluun luodaan ensin sovelluspohja, joka sisältää sovellukseen liitetyn versionhallintarepositorion ja konttirekisterin. Samassa yhteydessä luodaan myös levykuva käyttöjärjestelmästä, joka tulisi asentaa reunalaitteelle. Sovelluspohjan luonti voidaan mieltää reunalaitteiden ryhmittämisenä. Käyttöjärjestelmä sisältää luodusta sovelluksesta yksilöiviä tietoja, jotta reunalaitteen olisi mahdollista muodostaa automaattisesti VPN-yhteys keskitettyyn palveluun asennuksen ja käynnistämisen jälkeen.

Ohjelmiston toimittamisen keskiössä on versionhallintarepositorio. Kun ohjelmistoon tehdyt muutokset julkaistaan repositorion keskitetylle palvelimelle gitin *master*-haaraan, Balenan palvelu kääntää ohjelmiston, kokoaa kontin kuvan ja tallentaa sen konttirekisteriin. Tämän jälkeen reunalaitteille välitetään tieto ohjelmiston uudesta versiosta, jolloin konttien hallintaohjelmisto voi ladata sen käyttöönsä.

Reunalaitteiden ja Balenan palvelinten välille muodostetaan VPN-yhteys, jota käytetään välittämään tietoa uusista ohjelmistopäivityksistä. Reunalaitteiden toimintaa voi valvoa Balenan tarjoaman web-käyttöliittymän avulla. Balena on ensisijaisesti esineiden internetin infrastruktuurin alusta. Muita palveluita, kuten esineiden keräämän tiedon analysointia, Balena ei tarjoa. Balenan käyttölittymän avulla voi kuitenkin valvoa ohjelmiston toimintaa laitekohtaisesti.

4.6 Palveluntarjoajien ratkaisujen vertailua

Yhteenveto palveluiden käyttämien teknologioiden vertailusta on esitettynä taulukossa 1. Yleisellä tasolla AWS ja Microsoft Azure muistuttavat paljon toisiaan. Molemmat palvelut tarjoavat reunalaskentaan ratkaisuja, joissa yksittäistä ohjelmaa voidaan suorittaa joko periodisesti tai kutsuttaessa. Lisäksi molempien tarjoamiin ratkaisuihin kuuluu tekoälyalgoritmien suorittaminen reunalaitteissa. IBM:n Watson IoT -palvelu sisältää ratkaisuja ensisijaisesti esineiden ja niiden tuottaman tiedon hallintaan ja analysointiin. Microsoft Azuren ja AWS:n palvelut tarjoavat suoran keskitetyn ratkaisun sovellusten toimittamiseen reunalle. Watson IoT ei kuitenkaan tarjoa toimittamiseen suoraa ratkaisua.

Teknisesti ottaen Microsoft Azuren ratkaisut eroavat AWS:n ratkaisuista. Microsoft Azuren reunalaitteissa sijaitsevat moduulit, eli ohjelmistot toimitetaan konteissa.

Taulukko 1: Palveluntarjoajien ratkaisujen yhteenveto

Palvelu	**Palvelin-reuna -yhteys**	**Sovelluksen alusta**
IBM Watson IoT	MQTT	Kontit (IoT Edge)
AWS IoT	MQTT	Reunal. valmiiksi asennettu
Microsoft Azure Edge	MQTT/AMQPP	Kontit
Balena	VPN	Kontit

AWS:n toimitettavat yksiköt puolestaan sisältävät vain ohjelmakoodin, mikä edellyttää, että suoritusympäristö on valmiiksi asennettuna reunalaitteelle. AWS:n toimitettavat yksiköt voivat tällä tavoin olla pienempikokoisia ja nopeammin toimitettavia. Sekä AWS:n että Microsoft Azuren ratkaisut tukevat vain tiettyjä ohjelmointikieliä.

Balenan kontteihin perustuva ratkaisu mahdollistaa monipuolisten sovellusten kehittämisen reunalaitteisiin. Käyttäjän näkökulmasta sovellusten toimittaminen Balenalla on yksinkertaisinta verrattuna muiden tarkasteltujen palveluiden ratkaisuihin. Toimittaminen aloitetaan julkaisemalla ohjelmistoon tehdyt muutokset keskitetylle versionhallintapalvelimelle. Balenan tarjoamat palvelut tekevät muun tarvittavan automaattisesti. Balena tarjoaa pohjakuvia useille erilaisille sovellusten suoritusympäristöille Docker Hubissa.

Tarkastellut palvelut sisältävät mahdollisuuden tarkkailla järjestelmään liitetyn laitteen tilaa. AWS ja Balena tarjoavat tilantarkistuksen lisäksi mahdollisuuden lukea suoritettavan sovelluksen lokia. Microsoft Azure, AWS ja Watson IoT käyttävät pilvipalvelun ja reunalaitteen väliseen kommunikointiin pääosin MQTT:tä. Balena välittää viestejä palvelimelta reunalaitteille VPN-yhteyden avulla. Itse konttien lataamisessa käytetään HTTPS-protokollaa. Kontteja hyödynnetään sekä Balenan että Microsoft Azuren reunalaskentaratkaisuissa.

5 Aiempaa tutkimusta konteista reunalaskennassa

Tässä luvussa esitetään tutkimusta konttien soveltuvuudesta reunalaskentaan. Tarkastelussa huomioidaan erityisesti tapaustutkimuksia, joissa on pyritty hallinnoimaan reunalaitteita kontteja käyttäen. Tutkimusmateriaalin hakemiseen valittiin alunperin kaksi pääteemaa: kontit reunalaskennan ja esineiden internetin sovellusalustoina, sekä konttien toimittaminen reunalaskennassa. Artikkelien etsimisessä hyödynnettiin **IEEExplore Digital Library**[24] ja **The ACM Digital Library**[25] -tietokantoja. Hauissa käytettiin avainsanoja ”Internet of things”, ”delivery”, ”deployment”, ”containers”, ”edge computing”, ”fog computing”, ”docker” ja ”continuous”. Lähteiden valinnassa painotettiin erityisesti niiden tuoreutta ja relevanssia tutkimusaiheen kannalta. Tarkastelua laajennettiin iteratiivisesti hakutuloksista tunnistettujen keskeisten teemojen perusteella. Lopulliseen rajaukseen sisällytettiin kahden alkuperäisen teeman lisäksi konttiklusterien hyödyntäminen reunalaskennassa ja kontteihin liittyvä optimointi.

5.1 Konttien vaikutus sovellusten suorituskykyyn

Koska kontit ovat eräänlaista virtualisointia, vaikkakin kevyempää virtuaalikoneisiin verrattuna, herää kysymys niiden vaikutuksesta suoritettavan ohjelmiston suorituskykyyn. Kysymys koskee erityisesti laitteita, joiden resurssit ovat rajoitetummat kuin pilvilaskennan alustojen. Docker-konttien vaikutus sovellusten suorituskykyyn ja laitteiden virrankulutukseen on todettu olevan hyvin marginaalista korttitietokoneissa. Tämä tulos on saatu suorituskykyä mittaavista tutkimuksista, joissa saman ohjelman kontissa tehtyä suoritusta verrattiin natiivissa ympäristössä tapahtuneeseen suoritukseen. [33, 34]. Kokeet ovat käsitelleet Raspberry Pin lisäksi muita kuluttajamarkkinoilla saatavilla olevia samankaltaisia tuotteita. Raspberry Pin laskentatehon on todettu riittävän myös usean kontin samanaikaiseen käyttöönottoon [35]. CoAP-palvelinten (*Constrained Application Protocol*) suorituskykyä on mitattu Raspberry Pissa suoritetuissa konteissa [36]. Mittausten tulokset osoittivat Raspberry Pin soveltuvan CoAP-palvelinten suorittamiseen.

Tässä yhteydessä on syytä mainita Hypriot-projekti, joka on kevyt Raspberry Pille suunniteltu Debian-pohjainen käyttöjärjestelmä. Siinä on Dockerin ilmaisversio val-

[24] https://ieeexplore.ieee.org/Xplore/home.jsp
[25] https://dl.acm.org/

miiksi asennettuna[26]. Dockerin ilmaisversion asennuspaketteja on kuitenkin saatavilla muillekin Linux-pohjaisille käyttöjärjestelmille, jotka on suunniteltu Raspberry Pita silmällä pitäen. Yhteevetona voidaan todeta, että Docker soveltuu suorituskyvyltään reunalaitteiden konttiteknologiaksi. Jotta konttien mahdollisuudet voitaisiin hyödyntää monipuolisemmin, suorituskyvyn lisäksi on arvioitava niiden keskitetyn toimittamisen mahdollisuutta reunalaitteisiin.

5.2 Konttien toimittaminen reunalaskennassa

Toimittamismenetelmiä pohdittaessa konttikuvien rekisterin periodinen tarkastus uusien versioiden varalta on eräs vaihtoehto. Tätä on ehdotettu [37] ja se on hyvin mahdollinen ratkaisu pienessä mittakaavassa, esimerkiksi muutaman laitteen kokonaisuudessa. Jos laitteita on kuitenkin useita maantieteellisesti laajalla alueella, toimittaminen vain yhdelle laitteelle tai osalle laitteista on vaikeaa yksinkertaista periodista tarkastusta käyttäen. Tällaisessa operointitapauksessa jokaiselle laitteelle tai laiteryhmälle voidaan luoda konttirekisteriin oma repositorio. Jos ohjelmisto on sama jokaisessa ryhmässä tämä ei kuitenkaan ole mielekästä. Lisäksi sovellusten toimittamisen prosessia on vaikeaa valvoa, ellei laitteeseen ole yhteyttä. On pyrittävä hallinnoimaan yksittäisiä laitteita tarkemmin.

Toinen ratkaisu toimittamiseen on toteuttaa konttien hallinnoinnin toiminnot sisältävä web-pohjainen rajapinta jokaiseen laitteeseen [20]. Käyttäjä kutsuisi rajapintaa lataamaan ja ottamaan käyttöön kontteja tavallisilla HTTP-kutsuilla. Tällainen ratkaisu olisi mahdollinen vain, jos päivittävällä taholla on yhteys laitteeseen. Laitteen olisi itse muodostettava yhteys hallinnoivaan palvelimeen, jotta kahdensuuntainen kommunikaatio onnistuisi.

Foggy on eräs konttien resurssienhallintaan suunniteltu kehys reunalaskennan sovellusten toimittamiseen [38]. Sen tavoitteena on luoda kokonainen sovelluksen jatkuvan käyttöönoton putki versionhallintaohjelmistoa, integraatiopalvelua ja konttirekisteriä hyödyntäen. Tapaustutkimuksessa Foggya varten luotiin hallinnointipalvelin, joka valvoi esineiden ja yhdyskäytävien käyttämiä resursseja konttiklusterissa. Kokonaisuus jaettiin pilvilaskentaan, sumuun ja reunaan. Sumu- ja reunalaitteet kommunikoivat konttien hallinnointipalvelimen kanssa MQTT-protokollaa käyttäen. Tämä mahdollisti konttien asentamiseen, hallinnointiin ja monitorointiin tarvittavan rajapinnan etähallinnan. Artikkelissa esitetyt tulokset osoittivat konseptin ole-

[26]https://blog.hypriot.com/

van mahdollinen. Tutkimuksessa esitetty sovellusten toimittamisen kokonaisuus sisälsi versionhallinta- ja automaatiopalvelut, sekä konttien hallinnointiin luodun ohjelmistokokonaisuuden. Tämä osoittaa, että konttien suorittamisen lisäksi ketterien ohjelmistoprosessien käytännöt ovat mahdollisia reunalaskennassa.

5.3 Konttiklusterit reunalaskennassa

Sekä yksittäisten että ryhmitettyjen laitteiden konttienhallintaan tarvittavat toiminnot ovat toteutettuina konttien klustereita hallinnoivissa ohjelmistoissa. Docker Swarm sisältää useita tällaisia ratkaisuja, joita on potentiaalista hyödyntää reunalaskennassa. Konttiklusterit ovat suhteellisen monimutkaisia teknologioita. Niiden sisältämillä toiminnoilla voi olla laajempia vaikutuksia konttien suorituskykyyn. On siis arvioitava Docker Swarmin tai muiden konttiklusteriteknologioiden sopivuus pilvipalveluiden ja reunalaitteiden yhteenliittämiseen.

Docker Swarmia hyödyntävää konttien käyttöönottoa reunalaitteissa kokeiltiin tutkimuksessa, jonka infrastruktuuri koostui tietokeskuksesta ja reunalaitteista [39]. Docker Swarm -klusteri sijaitsi kokonaisuudessaan vain reunalaitteiden kerroksessa. Tietokeskuksessa sijaitsevaa rajapintaa käytettiin konttien siirtämiseen reunalla sijaitsevaan Docker-rekisteriin, josta Docker Swarmissa suoritettavat kontit otettiin käyttöön. Arvioitavina kriteereinä olivat toimittaminen, resurssien ja palvelun hallinnointi, virheiden sieto ja välimuisti. Tulokset osoittivat Docker Swarmin käytön mahdolliseksi reunalaskennan sovellusten toimittamisessa.

Docker Swarmin hallinnointilaitteen sijoittamista pilvipalvelimelle pohdittiin Muhammad Alamin ja kumppaneiden artikkelissa [40]. Järjestelmän kokonaisuus jaettiin kolmeen laskentakapasiteetiltaan erilaiseen kerrokseen. Tehokkain kerros, eli pilvikerros koostui laitteista, joihin sijoitettiin tiedon pitkäaikainen säilytys ja sen analysoinnin toiminnot. Keskimmäinen sumukerros koostui älykkäistä yhdyskäytävistä, joihin sijoitettiin hajautetun tiedonkäsittelyn ja -tallennuksen toiminnot. Viimeinen reunakerros koostui älykkäistä esineistä, jotka toimivat käyttöliittyminä käyttäjille. Kokonaisuuden hallinnoinnissa hyödynnettiin Docker Swarmia, jonka hallinnointilaitteet sijoitettiin pilvikerrokseen. Tekijälaitteet sijaitsivat sekä sumu- että reunakerroksessa. Artikkelissa kuvataan tapaustutkimus, jossa kolmesta Raspberry Pi -laitteesta koostettiin sekä reuna- että sumukerros. Pilvikerros koostettiin yhdestä kannettavasta tietokoneesta. Docker Swarmin hallinnointilaitteen sijoittaminen pilvikerrokseen ja tekijälaitteiden sijoittaminen reunaan ja esineisiin tarjoaa yksinkertaisen ratkaisun reunalaskennan ja esineiden internetin sovellusten toimittamiseen

ja muokkaamiseen. Docker Swarmin toimintoja voidaan tällöin hyödyntää suoraan esimerkiksi sovellusten jatkuvassa toimittamisessa.

Konttien hallinnointiin on esitetty Docker Swarmin sijaan myös Kubernetesia hyödyntäviä ratkaisuja [41, 42]. Kubernetesia, Mesosia ja Docker Swarmia vertailtiin yksinkertaisella vaatimusmäärittelyllä [43]. Eräs vaatimuksista oli lisälaitteiden liittäminen kontteja suorittaviin reunalaitteisiin. Yksikään ratkaisuista ei täyttänyt tätä vaatimusta. Sensorien ja aktuaattorien lisääminen reunalaitteeseen voi koitua hankalaksi. Reunalaitteita voi kuitenkin suhteellisen helposti käyttää palvelusuuntautuneen ohjelmiston alustoina. Tällöin konttia suorittava laite toimisi käytännössä vain älykkäänä yhdyskäytävänä, johon varsinaiset esineet ottavat yhteyden esimerkiksi kevyen protokollan avulla. Tutkimuksessa todettiin Docker Swarmin olevan kevyin vaihtoehto Mesosiin ja Kubernetesiin verrattuna. Vastaavasti toisessa vertailussa Docker Swarmin on todettu olevan sekä laskentatehon että muistinkäytön kannalta kevyempi Kubernetesiin verrattuna [44].

Kubernetesin hyödyntämistä hajautetussa tiedon analysoinnissa on arvioitu empiirisessä kokeilussa, jossa luotiin kaksi ohjelmistoa suuren tietomäärän käsittelyyn [45]. Ohjelmistot kehitettiin Tensorflow-ohjelmistokehyksen avulla. Kubernetes-klusteria käytettiin Docker-konttien kanssa. Kubernetes mahdollisti sovellusten käyttöönoton keskitetysti. Artikkelista ei käy ilmi millä tasolla keskitetty palvelin sijaitsi. Pääpaino oli konttiklusterissa suoritettavan tiedon analysoinnissa. Tulokset kuitenkin osoittivat Dockerin vaikuttavan tiedon analysoinnin suorituskykyyn hyvin vähän.

Kubernetes-klusterissa on mahdollista käyttää erilaisia päällysteverkkoteknologioita. Eräs tutkimus vertasi, **Flannel**, **Calico**, **OVN** ja **Weave** -nimisten päällysteverkkoratkaisujen suorituskykyä toisiinsa [42]. Verrokkina oli ratkaisu ilman päällysteverkkoa. OVN:n todettiin vaikuttavan suorituskykyyn kaikkein vähiten. Siinä missä Kubernetesissa on mahdollista käyttää erilaisia päällysteverkkoratkaisuja, Docker Swarm tukee vain yhtä, muokattavaa ratkaisua. Lisäksi vertailua on tehty Docker Swarmin, Calicon ja Flannelin välillä mitaten vastausten viivettä ja TCP- ja UDP-yhteyksien suoritustehoa [46]. Calico osoittautui tehokkaimmaksi heti päällysteverkottoman ratkaisun jälkeen, kun taas Flannelin todettiin vaikuttavan suorituskykyyn eniten. Nämä tutkimukset eivät liity suoraan konttien soveltamiseen reunalaskennassa, mutta niiden tulokset ovat hyödyllisiä minkä tahansa konttiklusterin rakentamisessa.

Fogernetes hyödyntää Kubernetesia reuna- ja sumulaskennassa [41]. Sen toiminta on jaettu kolmeen kerrokseen: pilveen, sumuun ja reunaan. Jokainen kerros on

tarkoitettu erilaiselle sovellukselle. Pilvikerroksessa suoritetaan laskennallisesti vaativimmat tehtävät. Reunakerros puolestaan sisältää kaikkein pienimmät laitteet. Sumukerros sijoittuu reunan ja pilven väliin toimien palvelimena asiakkaille ja reunalaitteille. Fogernetesia arvioitiin tapaustutkimuksessa rakentamalla kameravalvontasovellus. Reunakerrokseen sijoitettiin kamera, jonka laitteena toimi Raspberry Pi. Sumukerrosta ja pilveä kuvasivat virtuaalikoneissa suoritetut palvelimet. Määrittelyyn lisättiin ohjelmiston suorituksenaikainen valvonta. Konttien käyttöönotto onnistui Kubernetesin avulla.

Yhdyskäytävien konfigurointiin tarkoitetun **Kura**-ohjelmistokehyksen topologian laajentamista on esitetty Docker Swarmin avulla [47]. Docker Swarm, Kubernetes ja Apache Mesos mainittiin mahdollisina hallinnointiteknologioina. Empiirinen osuus keskittyi kuitenkin Docker-konttien vaikutuksiin testauksessa käytetyn ohjelman suoritusaikoihin. Lisäksi vertailtiin konttien kokoamisessa käytettävien tallennusajureiden vaikutuksia suoritusaikaan. Tulosten perusteella voidaan jälleen todeta Docker-konttien olevan mahdollisia suoritusympäristöjä reunalaskennan sovelluksille.

Konttiklusteriteknologian soveltaminen reunalaskennan yhteydessä vaikuttaa lupaavalta. Operoinnin näkökulmasta on otettava huomioon sovelluksen toimittamisen ja valvonnan mahdollisuudet, joihin löytyy jo ominaisuuksia sekä Docker Swarmista että Kubernetesista. Edellä mainittujen tutkimusten perusteella voidaan päätellä, että konttien hallinnointi on mahdollista laajassa pilvipalvelun ja reunalaitteiden muodostamassa verkossa. Klusterien hallinnointityökalut, kuten Docker Swarm ja Kubernetes ovat valmiita ratkaisuja, joiden on todettu sopivan reunalaskennan käyttötapauksiin.

5.4 Konttien optimointi ja katsauksen yhteenveto

Docker-kontit ladataan tavallisesti keskitetyltä palvelimelta. Reunalaitteiden verkkoyhteys voi olla hidas, joten konttien lataamisen optimoinnille on tarvetta. On osoitettu, että Docker-konttien latausprosessia on mahdollista nopeuttaa [48]. Tutkimuksessa esitettyjä optimointiehdotuksia olivat peräkkäinen kuvien kerrosten lataaminen, monisäikeinen pakkauksen purkaminen ja ladatun kuvan datan putkittaminen siten, että lataus ja kuvan pakkauksen purkaminen tapahtuvat samanaikaisesti. Konttien toimittamista onnistuttiin nopeuttamaan nelinkertaisesti. Lähtökohtana oli optimointi Raspberry Pi -tietokoneelle, mutta se ei sulje pois optimoinnin soveltamista muille laitteille.

Aiemman tutkimuksen perusteella voidaan päätellä, että Docker-kontit ovat sopiva alusta reunalaskennan sovelluksille. Empiiristen tutkimusten myötä todetaan, että konttiklusterit, pääosin Docker Swarm ja Kubernetes, ovat varteenotettavia vaihtoehtoja toteuttamaan ohjelmistojen valvonnan ja jatkuvan toimittamisen ratkaisuja. Erilaisten suoritinarkkitehtuurien huomioon ottaminen reunalaskennan sovellusten toimittamisen käytännöissä jäi kuitenkin avoimeksi tutkimuksia arvioitaessa.

6 Tapaustutkimus

Käytännön näkökulman muodostamiseksi suoritettiin tapaustutkimus konttien toimittamisesta ja valvonnasta reunalaskennassa. Tavoitteena oli luoda näkymä konttien mahdollisuuksiin reunalaskennassa ja toteuttaa sovellusten automaattisen toimittamisen mahdollistava kokonaisuus reunalaitteisiin. Tässä otettiin huomioon ohjelmiston suorituksenaikainen valvonta. Materiaalina hyödynnettiin tutkimuksessa käytettävien ohjelmistojen dokumentaatiota ja niihin liittyviä blogeja.

6.1 Tavoitteet

Tapaustutkimuksessa pyrittiin luomaan yksinkertainen, mutta ohjelmistokehitysprojektille tyypillisiä piirteitä sisältävä sovelluksen toimittamisen prosessi. Ohjelmisto tuli suorittaa reunalaitteissa pilvipalvelun ulkopuolella ja sen tuli olla operoitavissa keskitetysti. Sovellusten suorittamisen ja toimittamisen lähtökohtana olivat kontit, joiden on todettu olevan sopiva teknologia pilvilaskennan sovellusten operointiin. Tavoitteena oli tunnistaa haasteita, joita sovellusten toimittamisessa ilmenee reunalaskennassa.

Ensimmäinen vaatimus oli ohjelmiston tuotantoonviennin automaatio. Tavoitteena oli luoda toimittamisputki, jossa kehittäjän tai operoinnin tekijän on mahdollista ottaa ohjelmisto käyttöön välittömästi muutosten hyväksymisen jälkeen. Sovelluksen valvonta oli toinen vaatimus. Ohjelmiston suorituksenaikainen käyttäytyminen on operoinnin kannalta tärkeää niin ohjelmointivirheiden kuin käytön analysoinnin kannalta. Kolmas vaatimus oli ohjelmiston päivityksen peruuttaminen ennakoimattomassa virhetilanteessa. Sovellukseen saattaa päätyä ohjelmointivirheitä, jotka vaativat välitöntä käsittelyä.

Edellä mainitut vaatimukset ovat tyypillisiä palvelusuuntautuneen ohjelmiston kehityskäytäntöjä. Näillä vaatimuksilla arvioidaan toimittamisen prosessin sopivuutta reunalaskennan sovelluksiin. Reunalaskennan sovellusten toimittamisessa on huomioitava lisävaatimuksia suhteessa ainoastaan pilvilaskennassa suoritettaviin sovelluksiin.

Reunalaitteiden mahdollinen sijoittaminen maantieteellisesti laajalle alueelle tulee huomioida. Useita erilaisia käyttötapauksia toteuttava kokonaisuus voi sisältää laitteita erilaisiin tarkoituksiin. Laitteet voidaan esimerkiksi jakaa käyttöliittymän tarjoaviin ja sensoritietoa kerääviin. Tällöin jokainen laite voi suorittaa yksittäistä teh-

tävää. Toisin sanoen neljäntenä vaatimuksena oli laitteiden ryhmittäminen niin, että ohjelmistojen toimittaminen tiettyihin laitteisiin on mahdollista.

Esineiden internet on eräs reunalaskennan käyttötapauksista. Tämä herättää kysymyksen konttien mahdollisuudesta liittää sensoreita ja aktuaattoreita suoraan reunalaitteeseen oheislaiteliitäntöjä käyttäen. Dockerin avulla on mahdollista liittää tiedostojärjestelmän osia kontteihin tai suorittaa kontteja etuoikeutetussa (engl. *privileged*) tilassa niiden sisältämien toimintojen mahdollistamiseksi. Laitteiden liittäminen ei kuitenkaan ole ehdoton vaatimus, sillä reunalla suoritetut palvelut riittävät useisiin käyttötapauksiin.

Reunalaitteiden erilaisuuden vuoksi on huomioitava mahdollisuus rakentaa kokonaisuuksia arkkitehtuuriltaan erilaisista laitteista. Tarpeen vaatiessa ARM-suoritinten ja x86-suoritinten tulee pystyä suorittamaan samaa ohjelmistoa samaan klusteriin lisättyinä. Tämä mahdollistaa esimerkiksi laitteiston vaihtamisen portaittain, ja kattaisi sellaiset tapaukset, joissa tiettyä laitteistoa ei ole saatavilla tarpeeksi tai se puuttuu joltain maantieteelliseltä alueelta. Kuten sensorien ja aktuaattorien liittämisen tapauksessa erilaisten arkkitehtuurien sekoittaminen ei ole tässäkään ehdoton vaatimus. Tarkastelu sisällytettiin tutkimukseen ensisijaisesti Dockerin mahdollistaman moniarkkitehtuurisen konttikuvan testaamisen vuoksi.

Edellä mainitut vaatimukset esitetään muodollisesti seuraavassa listassa. Ne luetellaan järjestyksessä tärkeimmästä vähiten tärkeään.

1. Sovelluksen tulee olla toimitettavissa keskitetysti.
2. Sovelluksen ja laitteiden tulee olla valvottavissa.
3. Sovelluksen edellisen version tulee olla palautettavissa välittömästi.
4. Sovelluksen toimittamisen on onnistuttava tiettyyn laitteeseen tai laiteryhmään.
5. Sensoreiden ja aktuaattorien liittämisen laitteisiin tulee olla mahdollista.
6. Sovelluksen tulee olla toimitettavissa laitejoukkoon, joka koostuu arkkitehtuuriltaan erilaisista suorittimista.

Kirjallisuuden ja teollisuuden ratkaisujen tarkastelun pohjalta konteissa suoritettavien sovellusten toimittamiseen on havaittavissa kolme lähestymistapaa. Ensinnäkin uuden konttikuvan version tarkastus voi tapahtua periodisesti [37]. Toinen

mahdollisuus on erikoistunut sovellus. Se hallinnoi konttien toimittamisen ja valvonnan rutiineja palvelin- ja asiakasohjelmiston avulla reunalaitteen muodostamaa kaksisuuntaista kommunikaatioyhteyttä käyttäen [38, 20]. Kolmas tapa on hyödyntää konttiklusteriteknologiaa, jossa pilvessä sijaitsevaan klusteria hallitsevaan laitteeseen liitetään reunalla sijaitsevat tekijälaitteet.

Ensimmäinen lähestymistapa on yksinkertaisin toteuttaa. Monet käyttöjärjestelmäpaketit sisältävät valmiiksi periodiseen tehtävään tarkoitetut ohjelmistot. Uusien konttien lisääminen tai vanhojen poistaminen vaatii esimerkiksi ssh-yhteyden muodostamisen reunalaitteen ja hallinnoivan laitteen välille. Tällöin toimittamisen vaatimuksista ainoastaan ensimmäisen voidaan ennustaa toteutuvan. Muiden vaatimusten, kuten keskitetyn valvonnan toteuttamiseksi on otettava käyttöön lisäohjelmistoja.

Toinen lähestymistapa mahdollistaa monipuolisempia toimintoja. Kahdensuuntaisen kommunikaation mahdollistamiseksi reunalaitteen on muodostettava yhteys toimittavaan palvelimeen, esimerkiksi MQTT-sovellusprotokollaa hyödyntäen. Tällöin päivittävä käsky voidaan lähettää palvelimelta verkon reunalle. Erikoistuneen ohjelmiston voi rakentaa Dockerin ohjelmointirajapintaa hyödyntämällä. Erikoistuneita ohjelmistoja on kehitetty niin teollisuudessa kuin tutkimuksessa. VPN-yhteyden avulla voidaan myös muodostaa samankaltainen kommunikaatio, kuin Balenan ratkaisussa.

Kolmas vaihtoehto, eli konttiklusterin käyttäminen Docker Swarmin tai Kubernetesin avulla soveltuu sovellusten toimittamiseen laitteisiin. Päällysteverkon avulla on mahdollista kommunikoida palvelimelta reunalaitteeseen, jolloin sekä sovellusten toimittaminen että laitteiden valvonta toteutuvat samalla kertaa. Teollisuudessa kahdensuuntainen kommunikointi uuden ohjelmistoversion julkaisusta on ratkaistu kevyttä protokollaa käyttämällä. Konttiklustereiden käyttö on todettu empiirisissä tutkimuksissa mahdolliseksi reunalaskennan sovelluksissa. On siis syytä tehdä lisätutkimusta konttiklusterien potentiaalista reunalaskennan sovellusten operointiin. Konttiklusterit eivät ole sidottuja tiettyjen palveluntarjoajien tuotteisiin, vaan niiden käyttöönottoon riittää palvelin ja laitteet.

Tapaustutkimuksen kohteeksi valittiin Docker Swarm, sillä sen ratkaisu poikkeaa pilvipalveluntarjoajien valmiista tuotteista. Docker Swarm on yksinkertaisin konttiklusterivaihtoehto verrattuna Apache Mesosiin ja Kubernetesiin. Se kuuluu Dockerin vapaasti jaettavan version vakiotyökaluihin. Docker Swarmin valitsemista tukee lisäksi sen kuluttamien laskentaresurssien vähäisyys Kubernetesiin verrattuna [44].

Vaikka konttiklusterien käyttöä reunalaskennan välineenä on tutkittu, operoinnin näkökulma on jäänyt vähälle huomiolle. Tällä tutkimuksella paikataan löydettyä aukkoa.

Koska tapaustutkimuksessa sovellettavaksi teknologiaksi valittiin Docker Swarm, odotukset tuloksista olivat positiiviset ennen kokeen suorittamista. Suurimmat odotukset kohdistuivat vaatimusten 1, 2 ja 3 täyttämiseen, koska ne käsittelevät tavallisia tapauksia palveluohjelmistojen toimittamisessa. Odotukset vaatimuksen 4 täyttymiselle olivat myös suhteellisen positiiviset, sillä ryhmittelyn todettiin olevan mahdollista Docker Swarmin dokumentaation pohjalta.

Vaatimuksen 5, eli sensoreiden ja aktuaattorien liittämisen odotettiin lähtökohtaisesti täyttyvän, vaikka aiemmassa tutkimuksessa laitteiden liittäminen todettiin estetyksi Docker Swarmissa [43]. Dockerin kontteja on mahdollista suorittaa siten, että niillä on pääsy lisälaitteisiin. Tätä haluttiin kokeilla empiirisesti. Empiriaa motivoi lisäksi avoimen lähdekoodin ohjelmistojen, pääasiassa Docker Community Editionin kehitys. Aiemman tutkimuksen negatiivisista tuloksista huolimatta empiirinen kokeilu katsottiin aiheelliseksi, sillä Docker Community Editionia kehitetään jatkuvasti.

Vaatimuksen 6, eli sovellusten toimittamisen usean eri arkkitehtuurin suorittimista koostuvaan laitejoukkoon, todettiin olevan vähiten tutkittu aihe. Tällaisista tapauksista ei löytynyt tieteellisiä artikkeleita, joten arviointi perustuu kokonaan Dockerin dokumentaatioon ja empiiristen kokeilujen havaintoihin. Lähtökohdaksi otettiin Dockerin *manifest*-työkalu, joka mahdollistaa useita arkkitehtuureja tukevien konttien luomisen.

Seuraavassa aliluvussa esitetään empiirisen tapaustutkimuksen tekninen toteutus aloittaen yleiskuvauksella järjestelmästä, jatkaen ohjelmiston toteuttamiseen ja konttikuvan koonnin yksityiskohtiin. Painotus oli sovellusten operoinnissaa, eli toimittamisessa ja valvonnassa, joten toimitettava ohjelmisto tehtiin hyvin yksinkertaiseksi.

6.2 Toteutus

Tässä aliluvussa esitetään tapaustutkimuksessa toteutetun toimittamisputken tekninen kuvaus. Ensin käydään läpi tutkimuksessa käytetty laitteisto ja verkon topologia. Tämän jälkeen kuvaillaan toimitettavan sovelluksen toteutusta ja konttikuvan koonnin prosessia. Lisäksi puretaan Dockerin moniarkkitehtuurisen konttikuvan luominen, jota hyödynnettiin erilaisista arkkitehtuureista koostuvan klusterin kokeessa. Viimeiseksi kerrotaan konttikuvan toimittamisen käytännöstä.

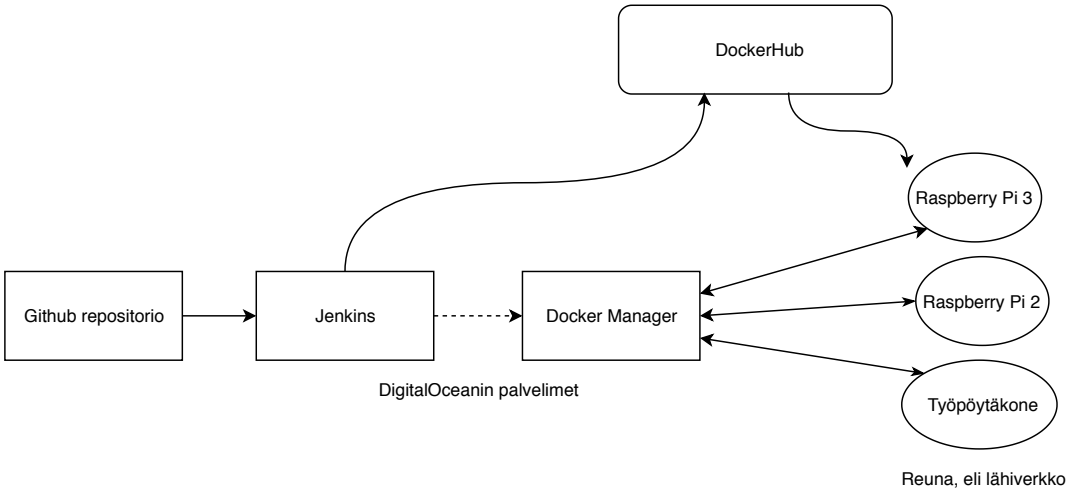
6.2.1 Tekninen yleiskuvaus

Sovellusten toimittamista varten luotiin automatisoitu toimittamisputki, joka koostui versionhallinnasta, automaatiopalvelusta ja Docker Swarm -klusterista. Klusteri koostui yhdestä hallinnoivasta laitteesta ja kahdesta hieman toisistaan poikkeavasta Raspberry Pi -korttitietokoneesta. Moniarkkitehtuurisessa kokeilussa oli lisäksi mukana yksi työpöytätietokone. Toimittamisen kohde, eli klusterissa suoritettava sovellus kehitettiin paikallisesti erillisellä koneella. Tällä tavoin luotiin tyypillisistä ketterän ohjelmistotuotannon käytännöistä koostuva prosessi.

Docker Swarmin hallinnoiva laite sijoitettiin pilvipalveluun. Tekijälaitteet, eli reunalaitteet sijoitettiin paikalliseen lähiverkkoon. Ensin otettiin ssh-yhteys reunalaitteeseen lähiverkossa. Tämän jälkeen kyseinen laite liitettiin hallinnoivan laitteen Docker Swarm -verkkoon manuaalisesti. Tekijälaitteet koostuivat Raspberry Pi 2 B ja Raspberry Pi 3 B+ -korttitietokoneista, joihin oli asennettu käyttöjärjestelmäksi Raspbian Lite. Docker Community Edition asennettiin molempiin laitteisiin erikseen. Raspberry Pi 2:n suorittimena toimi neliytiminen ARM Cortex-A7 900 megahertsin kellotaajuudella. Raspberry Pi 3:n suoritin oli 1,4 gigahertsin kellotaajuudella toimiva Cortex-A53. Molemmissa on 1 gigatavu keskusmuistia. Raspberry Pi 3:ssa oli lisäksi vakiona langattoman verkon adapteri ja mahdollisuus käyttää bluetoothia. Raspberry Pi 2 liitettiin langattomaan lähiverkkoon erillisen USB-adapterin avulla.

Arkkitehtuuriltaan erilaisista laitteista koostuvan klusterin kokeessa hyödynnettiin pöytätietokonetta, jonka suorittimena oli 3,2 gigahertsin kellotaajuudella toimiva Intelin i5 2500K. Koska kyseinen suoritin poikkeaa sekä teholtaan että käskyarkkitehtuuriltaan paljon molemmista Raspberry Pi -korttitietokoneista, molemmille koneille oli koottava erilliset Docker-kuvat.

Jenkins-automaatiopalvelu ja Docker Swarmia hallinnoiva laite suoritettiin omilla



Kuva 4: Yleiskuva toteutuksen rakenteesta.

virtuaalipalvelimillaan, jotka vuokrattiin DigitalOcean-palvelusta[27]. Molemmat virtuaalipalvelimet sisälsivät Ubuntu 18.04 -käyttöjärjestelmän. Jenkinsiä suorittavalle palvelimelle varattiin 25 gigatavua tallennustilaa ja 2 gigatavua keskusmuistia. Docker Swarmin hallinnoiva laite sisälsi vain yhden gigatavun keskusmuistia, mutta muuten kokoonpano oli sama kuin Jenkinsin palvelimessa. Sekä reunalaitteille että virtuaalipalvelimille asennettiin Docker Community Editionin versio 18.09.

Kuvassa 4 esitetään tapaustutkimuksen tekninen toteutus yleisellä tasolla. Kahdensuuntaiset nuolet esittävät klusterin hallinnointipalvelimen yhteyttä reunalaitteisiin. Yhdensuuntainen nuoli Jenkins-palvelimelta DockerHubiin esittää kuvan lataamista konttirekisteriin. Nuoli DockerHubista reunalle ilmentää konttikuvan lataamista reunalaitteille. Katkoviiva Jenkinsin ja Docker Managerin välillä kuvaa ssh-yhteyttä, jota hyödynnettiin klusterin käskyttämisessä. Docker Swarmia käytettiin perusasetuksin. Klusterin käyttämän päällysteverkon oletusasetuksia ei muutettu, eikä muita erillisiä päällysteverkkoja käynnistetty.

6.2.2 Sovelluksen tekninen toteutus ja konttikuvan koonti

Reunalaitteen sovellus toteutettiin Go-ohjelmointikielellä (Liite 1). Sovellus käännettiin pienikokoisiksi binäärimuotoisiksi tiedostoiksi. Niiden suorittamiseen ei vaadita erillisiä ohjelmointikielten suoritusympäristöjä, jotka saattavat viedä paljon levytilaa. Tällöin Docker-kuvat oli mahdollista pitää hyvin pieninä, ottaen huomioon verkkoyhteyden hitauden. Gon ohjelmat voidaan myös kääntää usealle suoritinark-

[27]https://www.digitalocean.com/

```
docker build . -f Dockerfile.arm -t kayttajanimi/ohjelma:arm
docker build . -f Dockerfile.x86 -t kayttajanimi/ohjelma:x86
docker push kayttajanimi/ohjelma:arm
docker push kayttajanimi/ohjelma:x86
docker manifest create kayttajanimi/ohjelma:latest \
  kayttajanimi/ohjelma:arm \
  kayttajanimi/ohjelma:x86
docker manifest push kayttajanimi/ohjelma:latest
```

Listaus 3: Moniarkkitehtuurisen kuvan koonti.

kitehtuurille. Itse sovellukseen toteutettiin vain yksittäinen GET-metodilla kutsuttava WEB-polku, joka palautti *JSON*-muotoisen HTTP-vastauksen. Ohjelmiston päivittämistä kokeiltiin muuttamalla kyseisen vastauksen muotoa ja palautettavaa tietoa.

Dockerin kuville on mahdollista toteuttaa **monivaiheinen koonti** (engl. *multi-staged build*). Tämä toiminto lisättiin Dockerin versioon 17.05. Monivaiheisen koonnin ajatuksena on, että sovelluksen lähdekoodi voidaan kääntää erilaisessa ympäristössä, kuin missä se lopulta suoritetaan. Monivaiheisessa koonnissa voidaan käyttää useita pohjakuvia. Golla kirjoitettu käännetty binäärimuotoinen sovellus saattaa olla kooltaan kymmenen megatavun luokkaa, jolloin olisi käytännöllisintä olisi lopullinen suoritettava Docker-kuva kooltaan pienikokoiseksi.

Kuvan koonnin vaiheet jaettiin kahteen osaan: sovelluksen kääntämiseen ja lopullisen sovelluksen käynnistämiseen. Koonnin ensimmäisessä vaiheessa käytetty Go-kääntäjän sisältämä pohjakuva on Dockerin virallisesti ylläpitämä. Go-kääntäjän lisäksi se sisältää useita tyypillisiä Unix-käyttöjärjestelmien työkaluja. Nämä ovat tarpeettomia lopullisen sovelluksen suorittamisessa. Kuva oli kooltaan yli 800 megatavua. Toisessa vaiheessa oli otettava huomioon sekä suorittimen arkkitehtuuri että lopullisen kuvan koko. Sovelluksen suorittamisen kuvaksi valittiin *alpine*, joka on pienin mahdollinen pohjakuva. Docker Hubista on saatavilla pohjakuvia erilaisia arkkitehtuureja varten. Kahta erilaista suoritinarkkitehtuuria ajatellen koontia varten luotiin kaksi erilaista Dockerfilea (Liitteet 2 ja 3). Koska sovelluksen ohjelmointikielenä käytettiin Gota, sovellus oli käännettävä erikseen ARM- ja x86-arkkitehtuureille. Lopullinen Docker-kuva oli kooltaan vain noin 11 megatavua, mikä sopi erinomaisesti toimittamiseen hitaammassakin verkossa.

Moniarkkitehtuurisen kuvan luomiseen hyödynnettiin manifest-työkalua. Manifestin ajatuksena on, että yhden Docker-kuvan tunniste, esimerkiksi *latest*, osoittaa useaan erilaiseen arkkitehtuuriin. Kukin Docker-asiakas valitsee itselleen sopivan

automaattisesti. Tällöin toimittamista ei tarvitse eritellä jokaiselle arkkitehtuurille erikseen. Moniarkkitehtuurisia kuvia luodaan siten, että jokaiselle arkkitehtuurille kootaan ensin sopiva Docker-rekisterissä julkaistava kuva. Kuvat voidaan julkaista samaan repositorioon asettaen niille eri tunnisteet. Tämän jälkeen manifest-työkalun avulla määritellään *manifest-lista*, johon liitetään aikaisemmin luodut kuvat. Luotu manifest-lista julkaistaan rekisteriin ladattavaksi. Listaus 3 sisältää konkreettisen esimerkin manifest-työkalun käytöstä. Esimerkissä kootaan ja julkaistaan moniarkkitehtuurinen konttikuva kahta eri Dockerfilea käyttäen. Dockerin dokumentaation mukaan Manifest on kokeellinen toiminto, joten sen käyttämistä ei vielä suositella tuotanto-ohjelmistoille.

6.2.3 Sovelluksen toimittaminen

Sovelluksen toimittaminen toteutettiin Docker Swarmin *stack*-toiminnon avulla. Toimittamisen automatisointiin luotiin Jenkins-palvelin, johon luotiin koontia ja toimittamista varten kaksi erillistä tehtävää. Ensimmäinen luotiin vaiheittaiseksi Pipeline-tehtäväksi. Jokainen vaihe oli suoritettava hyväksytysti, jotta seuraavan vaiheen suorittaminen voi alkaa. Vaiheet olivat järjestyksessä lähdekoodin lataaminen Githubista paikalliselle Jenkins-koneelle, Docker-kuvan kokoaminen ja lopulta sen lataaminen rekisteriin (Liite 4). Toinen tehtävä suoritti sovelluksen toimittamisen ssh-yhteyden avulla komentamalla konttiklusteria hallinnoivaa palvelinta päivittämään halutun palvelun. Sovelluksen toimittamisessa käytettiin *docker stack deploy* -komentoa, jolle syötettiin parametrina uusin versionhallinnasta löytyvä docker-compose.yml -tiedosto (Liite 5).

Pipeline-tehtävä määriteltiin tiedostoon *Jenkinsfile*, joka tallennettiin versionhallintaan. Prosessin muutokset pysyivät tällöin versionhallintajärjestelmän historiassa. Docker-kuvan koostaminen ja toimittaminen määriteltiin erillisiksi tehtäviksi, jotta päivityksen tekeminen onnistuisi ilman uutta Docker-kuvaa sovelluksesta. Tällä katettiin tapaukset, joissa pyritään muuttamaan ainoastaan suoritettavan kontin asetuksia ilman muutoksia suoritettavaan sovellukseen. Esimerkiksi kontin varaamaa muistin vähimmäismäärää voidaan muuttaa erikseen.

Taulukko 2: Vaatimusten tulokset

nro.	**Vaatimus**	**Docker Swarmin soveltuvuus**
1	Sovelluksen toimittaminen	Soveltuva
2	Monitorointi	Soveltuva
3	Edellisen version palautus	Soveltuva
4	Ryhmittäminen	Soveltuva
5	Sensorit & aktuaattorit	Ei soveltuva
6	Eri suoritinarkkitehtuurit	Soveltuva

6.3 Tulokset

Empiirisen tutkimuksen tulokset on tiivistetty taulukkoon 2. Niistä voidaan havaita, että Docker Swarm täytti lähes kaikki sovellusten toimittamiseen määritellyt vaatimukset. Docker Swarmia voi hyödyntää reunalaitteiden sovellusten toimittamisessa, eli vaatimus 1 meni läpi. Toimittamistapa ei eroa tavanomaisten palvelusuuntautuneiden ohjelmistojen toimittamisesta, joten konttiklusteri on mahdollinen teknologia räätälöityjen reunalaskennan ratkaisujen rakentamiseen. Toimittamisprosessin voi automatisoida suurilta osin. Laitteiden suorituskyvyn parantuessa tulevaisuudessa, voi olla mahdollista luoda jatkuvan toimittamisen ja käyttöönoton keskitetty automaattinen putki etäisissä lähiverkoissa sijaitseviin laitteisiin Docker Swarmin avulla. Uuden version toimittaminen ohjelmistosta onnistui. Tosin reunalaitteessa suoritettava ohjelmiston toiminnassa havaittiin lyhyt katko, kun uutta konttia otettiin käyttöön. Havainto tehtiin tekemällä toistuvia pyyntöjä ohjelmiston web-rajapinnalle selaimen avulla.

Vaatimus 2 oli reunalaskennan sovellusten keskitetty valvonta. Docker Swarm tarjoaa mahdollisuuden lukea suoritettavien sovellusten lokeja *docker service logs* -komennolla, joten vaatimus täyttyi. Palveluiden sovellukset on suunniteltava tuottamaan lokiviestejä toiminnastaan. Lokien toimintaa tutkittiin muodostamalla ssh-yhteys hallinnointilaitteelle komentoriviohjelmaa käyttäen ja suoraan käyttämällä hallinnointilaitteen Docker-ohjelmiston toimintoja. Tekijälaitteiden konttien resurssienkäyttöä ei kuitenkaan voi valvoa. Esimerkiksi kontin varaama muisti voi olla haluttu tieto. Tämä voi olla rajoite valvonnalle. Reunalaitteiden yhteyden tilaa voi kuitenkin seurata hallinnoivalta palvelimelta.

Docker Swarm tukee Rollback-ominaisuutta, jolla on mahdollista palauttaa edellinen

versio kontista käyttöön (vaatimus 3). Rollback-mahdollisuutta tarvitaan tavallisesti vain hyvin vakavien ennalta-arvaamattomien virheiden esiintyessä. Vaihtoehtona Docker Swarmin Rollback-toiminnolle olisi voinut olla konttien versiointi tunnisteiden avulla. Näin edellisen virheettömän version käyttöönottaminen tuotantoon olisi onnistunut käyttäen samoja toimintoja kuin päivittämisessä. Tällaisen menetelmän kehittäminen todettiin kuitenkin tarpeettomaksi rollback-toiminnon ollessa valmiiksi saatavilla. Rollback-toimintoa kokeiltiin empiirisesti muodostamalla ssh-yhteys hallinnointilaitteeseen komentoriviohjelmalla.

Vaatimusten 1, 2 ja 3 ennusteet olivat positiiviset. Toimittaminen, virhetilanteiden valvominen ja niihin reagoiminen ovat ensisijaisia toimintoja palveluiden operoinnissa. Docker Swarm vaikuttaa näihin hyvin soveltuvalta teknologialta reunalaskennassa. Seuraavassa esitetään vaatimuksia, joiden rooli korostuu reunalaskennan käyttötapauksissa.

Laitteiden ryhmittäminen erillisiin joukkoihin ja sovellusten toimittaminen laiteryhmiin onnistui (vaatimus 4). Docker Swarm tukee klusterin laitteiden ryhmittämistä leimoilla (engl. *label*), joiden avulla voidaan määritellä tarkasti, mihin palvelun suoritus sijoitetaan. Leimat ovat joko avain-arvo -pareja tai pelkkiä arvoja. Esimerkiksi reunalaitteille, joille toimitetaan viestinvälittäjäohjelmisto, voidaan asettaa leima *type=messagebroker*. Palvelua toimitettaessa on määriteltävä rajoitteet (engl. *constraints*) palvelun suorittavan kontin sijoituspaikalle. Edellä mainitun viestinvälittäjäesimerkin mukaan rajoite määriteltäisiin syöttämällä Docker-asiakasohjelmalle joko komentoriviparametreina *"–constraints node.labels.type == messagebroker"* tai yml-muotoisessa *docker-compose*-tiedostossa. Jälkimmäistä käytettiin tapaustutkimuksessa. Konttien suorittamisen sijainnin voi rajoittaa yksittäisiinkin laitteisiin. Jokaisella klusterin laitteella on *"hostname"*-kenttä, jota voi käyttää sen tunnistamiseen. Hostname on määriteltävissä vain jokaisen laitteen käyttöjärjestelmän toiminnoista käsin. Sitä ei siis voi asettaa keskitetysti. Yksilöivän tunnuksen asettamisprosessiin on varauduttava hyvissä ajoin ennen kehityksen ja operoinnin alkamista, erityisesti jos reunalaitteita on useita. Leimojen asettamista ja ryhmittämistä tutkittiin ssh-yhteyden välityksellä tehdyillä komennoilla klusterin hallinnointilaitteessa. Leimoja lisättiin *docker node update* -komennolla.

Docker Swarmia käytettäessä ilmeni rajoitteita lisälaitteiden liittämisessä (vaatimus 5). Pelkkiä kontteja käytettäessä laitteiden kuvaaminen konttiin onnistuu, mutta Docker Swarm ei tue kuvaustoimintoja. Kontin liittämiseen lisälaitteisiin on kolme tapaa. Ensimmäinen vaihtoehto on konttia käynnistettäessä sitoa (engl. *bind*) tarvit-

tavat isäntäjärjestelmän laitteisiin osoittavat tiedostot Dockerin volyymeja hallinnoivalla toiminnolla. Toinen vaihtoehto on asettaa kontin käynnistyksen yhteydessä *privileged*-asetus, joka avaa kontille mahdollisuudet käsitellä kaikkia lisälaitteita. Kolmas vaihtoehto on liittää halutut laitteet *device*-parametrilla. kaikkia näitä asetuksia voidaan määritellä docker-compose -tiedostoon, mutta Docker Swarmia käytettäessä ne jätetään huomiotta. Voidaan esittää, että Docker Swarmin avulla on käytännöllisintä luoda ainoastaan palvelusuuntautuneita ohjelmistoja. Tämä tarkoittaa, että esimerkiksi USB-väylään liitettävien mittalaitteiden lisääminen tai Raspberry Pin GPIO-liittimien hyödyntäminen ei ole mahdollista. Docker Swarmin soveltaminen rajoitti käyttötapauksia. Sensoreita ja aktuaattoreita sisältäviä laitteita voi kuitenkin edelleen lisätä erillisinä laitteina samaan lähiverkkoon, jossa reunalaite sijaitsee. Tällöin tiedon siirtäen tapahtuu jotain tietoliikenneprotokollaa, kuten MQTT:tä hyödyntäen. Käyttötapauksissa, joissa lisälaitteita on liitettävä suoraan reunalaitteisiin, tulisi valita jokin muu vaihtoehto Docker Swarmin sijaan. Rajoitukset todettiin empiirisesti määrittelemällä lisälaitteiden liittämiseen tarvittavia asetuksia docker-compose -tiedostoon.

Vaatimus 6 koski sovelluksen toimittamista erilaisista suorittimista koostettuun klusteriin. Tämä todettiin mahdolliseksi manifest-työkalun avulla. Automaattinen toimittaminen ei kuitenkaan onnistunut Jenkinsiä käyttäen, sillä kyseisen ohjelmiston liitännäiset eivät tukeneet suoraan Dockerin kokeellisia ominaisuuksia. Toimittaminen onnistui kehittäjän paikalliselta laitteelta, jossa Dockerin kokeelliset ominaisuudet olivat saatavilla. Paikallista konttikuvien koontia varten tehtiin skripti, jota kutsumalla voitiin yksinkertaistaa sovelluksen konttikuvien luomisen prosessi (Liite 6). Skriptin suorituksen jälkeen kehittäjän oli kutsuttava Jenkinsistä toimittamistehtävää manuaalisesti.

7 Johtopäätökset ja pohdinta

Tässä tutkielmassa luotiin katsaus reunalaskennan ratkaisuihin. Konttien soveltamista reunalaskentaan käsiteltiin ketterien kehitysprosessien käytäntöjen näkökulmasta. Katsauksella nykyaikaisia teollisuuden ratkaisuja tarkastelemalla luotiin kuva tekniikoista, joita reunalaskennan sovellusten toimittamisessa sovelletaan. Empiirisillä kokeilla onnistuttiin rakentamaan yksinkertainen sovellusten toimittamisprosessi. Tämä tapahtui hyödyntäen pääosin samoja ratkaisuja, joita on käytetty palvelusuuntautuneiden ohjelmistojen toimittamisessa. Tässä luvussa esitetään vastaukset tutkimuskysymyksiin ja pohditaan mahdollisia jatkotutkimusaiheita.

7.1 Reunalaskennan sovellusten toimittamisen ratkaisut

Ensimmäinen tutkimuskysymys kuului: **Millaisia teknisiä ratkaisuja reunalaskennan sovellusten toimittamiseen on sovellettu?** Lyhyesti voidaan todeta teknisten ratkaisujen vaihtelevan käytettyjen tietoliikenneprotokollien osalta. Toimittamisen menetelmä on kuitenkin päällisin puolin hyvin samankaltainen riippumatta käytetystä teknologiasta. Tarkastelluista ratkaisuista jokainen sisälsi kokonaisen käyttöjärjestelmän käytön reunalaitteessa. Sivuhuomautuksena on mainittava, että IBM Cloudin tarjoama Watson IoT ei sisältänyt suoraa ratkaisua reunalaitteiden sovellusten keskitettyyn toimittamiseen ja päivittämiseen. Watson IoT:n ratkaisuja tarkasteltaessa kuitenkin löydettiin IBM:llä alkunsa saanut ohjelmistokehys **Node-RED**, joka arvioitiin potentiaaliseksi työkaluksi reuna- ja pilvilaskennan ohjelmistokokonaisuuksien rakentamiseen.

Palveluntarjoajat ratkaisivat sovellusten hallinnoinnissa tarvittavan palvelimen ja reunalaitteen välisen kommunikoinnin pääosin kahdella eri tavalla. Kommunikointimenetelminä käytettiin sekä kevyiden tietoliikenneprotokollien ratkaisuja, kuten MQTT:tä [38] että VPN-yhteyttä, kuten Balenan tarjoamassa palvelussa. Teollisuuden ratkaisuissa toimittamisen välineinä ja sovellusten suoritusympäristöinä oli sekä kontteja että reunalaitteissa suoritettuja kokonaisia ohjelmistoalustoja. Kontteja hyödyntävien ratkaisujen havaittiin käyttävän kaksivaiheista toimittamismenetelmää: ensin reunalaitteille välitetään tieto uudesta konttikuvasta, minkä jälkeen reunalaitteet lataavat uuden konttikuvan keskitetyltä konttirekisteripalvelimelta.

Dockerin havaittiin olevan suosituin konttiteknologia. Sen todettiin useassa tutkimuksessa olevan sopiva sovellusten suorittamiseen Raspberry Pilla ja vastaavilla korttitietokoneilla [33, 34, 35, 36]. Aiempien empiiristen tutkimusten ja ehdotettujen

ohjelmistokehysten perusteella voidaan päätellä, että Docker Swarm ja Kubernetes ovat sopivia konttien hallinnoinnin välineitä reunalaskennassa [39, 40, 47, 41, 42]. Yksi tutkimus koski reunalaitteissa sijaitsevien konttien hallinnointia web-rajapinnan avulla [20].

Keskitetty hallinnointi vaatii, että reunalaitteet muodostavat ensin yhteyden keskitettyyn palvelimeen. Balena ja AWS Greengrass tarjoavat menetelmän automaattiseen yhteyden muodostamiseen. Molempien terminologia kattaa laiteryhmät, joihin voidaan lisätä yksittäisiä laitteita. Operointia harjoittavan tahon tehtäväksi jää asentaa laitteelle ohjelmisto, joka sisältää automaattiseen yhteydenmuodostamiseen tarkoitetut asetukset. AWS Greengrassin ratkaisu sisältää yhteyden muodostamisen ohjelmiston, jota käyttöönotettaessa reunalaitteessa tulisi olla asennettuna jo kokonainen käyttöjärjestelmä. Balenan ratkaisu taas itsessään on erikoistunut käyttöjärjestelmä. Muiden tarkasteltujen palveluntarjoajien ratkaisut yhteyden muodostamiseen vaativat operoinnin harjoittajalta manuaalisia toimenpiteitä.

Tutkimuksen luotettavuuden kannalta on huomioitava, että katselmoinnissa käytettiin vain palveluntarjoajien vapaasti saatavilla olevia dokumentaatioita. Niissä ei aina suoraan mainittu ratkaisujen hyödyntämiä teknologioita. Tällöin tieto järjestelmän teknisestä toiminnasta jäi vähäiseksi. Lisäksi katselmointiin valittiin vain neljä erilaista palveluntarjoajaa, joista kolme oli pitkään alalla toimineita (Amazon Web Services, Microsoft Azure, IBM Cloud) ja yksi oli suhteellisen nuori yritys (Balena). Vaikka kohteita oli vain muutama, niiden tarjoamista sovellusten toimittamisen ratkaisuista saatiin muodostettua kuva. Teollisuuden ratkaisujen tarkastelua vahvistettiin käymällä läpi tutkimuksia.

Pilvipalveluntarjoajien, kuten Amazon Web Servicen ja Microsoft Azuren kehittämät ratkaisut laskennan siirtämiseksi lähemmäs reunaa osoittavat kiinnostuksen reunalaskentaan olevan liikemaailmassa suurta. Viime vuosien aikana kontit ovat haastaneet virtuaalikoneet sovellusten eristysteknologiana. Kevyt käyttöjärjestelmätason virtualisointi on avannut mahdollisuuksia keskitetyn operoinnin toteuttamiseen reunalaskennassa pilvilaskennan lisäksi. Microsoft Azuren ja Balenan konttiteknologiaa hyödyntävät ratkaisut osoittavat teollisuuden olevan kiinnostunut niiden käytännöistä reunalaskennassa palvelimilla suoritettavien palveluiden lisäksi. Teollisuuden tilanteen myötä on havaittavissa, että ketterien menetelmien käytäntöjen soveltamiseen reunalaskennassa pyritään aktiivisesti. Hajautettuihin reunalaitteisiin on mahdollista toimittaa ohjelmistoja samoja työkaluja käyttäen, kuin pilvipalveluissa sijaitseviin palvelimiin.

7.2 Konttiklusterien soveltuvuus reunalaskennan sovellusten operointiin

Toinen tutkimuskysymys kuului: **Voiko Docker Swarmia hyödyntää reunalaskennan sovellusten operoinnissa?** Empiirisen tapaustutkimuksen perusteella voidaan todeta, että Docker Swarmin avulla on mahdollista rakentaa jatkuvan toimittamisen putki reunalaskennan sovellusten keskitettyyn hallinnointiin. Docker Swarmia hyödyntäen onnistuttiin rakentamaan käytäntö, joka mahdollisti samoja toimintoja kuin teollisuuden ratkaisuissa. Tärkeimpinä toimintoina ohjelmisto kyettiin toimittamaan keskitetysti ja sen suorituksenaikaista käyttäytymistä voitiin valvoa.

Tapaustutkimuksessa ilmenneet haasteet ovat samansuuntaisia kuin aiemmissa tutkimuksissa esitetyt. Laitteiden hallinnointi keskitetysti ja niiden nimeäminen osoittautuivat haastaviksi [5]. Nimeämisongelma koski toimittamisen lisäksi reunalaitteissa suoritettavien ohjelmistojen tuottaman tiedon tunnistamista. Sovellusten toimittamisen haasteita esiintyi verkon hajanaisuudessa ja laitteiden heterogeenisyydessä [5, 19]. Tässä tutkimuksessa toimittamisen haasteet ilmenivät tarkemmin reunalaitteiden yhteyden muodostamisessa palvelimen ja laitteen välille, verkkoyhteyden nopeudessa ja toimitettavan ohjelmiston koossa, sekä kehitys- ja tuotantolaitteiden toisistaan poikkeavissa arkkitehtuureissa.

Varsinaista toimittamista suunniteltaessa haasteeksi todettiin ensinnäkin yhteyden muodostaminen reunalaitteen ja operointipalvelimen välille. Millä tavoin voidaan taata palvelimen viestien välittyminen reunalaitteelle sovellusta toimitettaessa? Tapaustutkimuksessa hyödynnettiin Docker Swarmin päällysteverkkoratkaisua, jonka todettiin toimivan kahdensuuntaisena kommunikaatiomenetelmänä. Eräs empiirisen kokeilun toimittamiseen liittyvistä vaatimuksista oli laitteiden ryhmittäminen. Tämän todettiin olevan suoraan yhteydessä nimeämisongelmaan ja laitteiden yksilöintiin [5]. Tapaustutkimuksessa nimeämisongelma ratkaistiin Docker Swarmin leimojen avulla.

Tapaustutkimuksessa toiseksi haasteeksi paljastui verkon nopeus. Ohjelmistoa kehitettäessä todettiin, että hitaiden yhteyksien huomioimisen kannalta on käytännöllisintä pyrkiä pitämään ohjelmiston koko mahdollisimman pienenä. Suurten konttien lataaminen voi viedä paljon aikaa ja resursseja. Lisäksi oli otettava huomioon reunalaitteen rajallinen muistikapasiteetti kehittämisessä käytettyyn laitteeseen verrattuna. Dockerin monivaiheista koontia hyödyntämällä onnistuttiin luomaan pieniko-

koinen sovellus, joka voitiin toimittaa nopeasti reunalaitteille.

Suoritinten arkkitehtuurien erilaisuus todettiin kolmanneksi haasteeksi. Jotta ohjelmiston toiminta voitiin taata sekä kehitys- että reunalaitteissa, oli muodostettava prosessi, jossa otettiin huomioon ohjelmiston koonti oikealle arkkitehtuurille. Lisäoletuksena otettiin huomioon, että reunalaitteet voivat koostua keskenään erilaisista laitteista. Empirian avulla Dockerin todettiin olevan soveltuva näiden haasteiden ratkaisuksi. Dockerin konttikuvia on mahdollista koota käyttäen erilaisia suorittimia. Lisäksi toisistaan poikkeavilla suorittimilla on mahdollista luoda kontteja, jotka toimivat arkkitehtuuriltaan erilaisissa suorittimissa.

Empiirisessä kokeilussa käytettiin Raspberry Pita, johon voi liittää lisälaitteita helposti. Näin ollen harkittiin konttien soveltuvuutta aktuaattorien ja sensorien hallinnointiin. Docker Swarm kuitenkin rajoitti käyttötapaukset lähinnä reunalla sijaitseviin palvelusuuntautuneisiin ohjelmistoihin. Lisälaitteiden liittäminen ei onnistunut. Rajoitukset voivat kuitenkin olla väliaikaisia, sillä Dockerin ohjelmistokokonaisuutta kehitetään jatkuvasti. Tulevaisuudessa on mahdollista, että reunalla sijaitseva älykäs yhdyskäytävä voi havainnoida ympäristöään suoraan sensoreiden avulla ja vaikuttaa siihen aktuaattorien kautta. Aiemmassa tutkimuksessa päädyttiin samankaltaisiin tuloksiin laitteiden liittämisen osalta [43].

Vaikka tämän tutkimuksen empiirinen osuus tuotti negatiivisen tuloksen sensoreiden ja aktuaattorien liittämisestä Docker Swarmia käytettäessä, niiden lisääminen voi olla mahdollista Kubernetesia käyttäen [41]. Teollisuuden ratkaisuista Balenan havaittiin suorittavan kontteja etuoikeutetussa tilassa, mikä mahdollistaa lisälaitteiden käytön. Erityisenä huomiona on mainittava, että laitteiden liittämisen ongelman voi kiertää luomalla erillisen palvelun, joka käynnistää Docker-asiakasta suorittavan kontin. Kontille sidotaan yhteys Dockerin liitäntään (engl. *socket*), joka sijaitsee Unix-pohjaisessa käyttöjärjestelmässä polussa */var/run/docker.sock*. Tällöin kontti voi käynnistää halutun laitteita käyttävän kontin etuoikeutetussa tilassa. Tämä vaikutti kuitenkin ongelman kiertävältä ratkaisulta, eikä pysyvältä tuotantoon sopivalta vaihtoehdolta. Erillinen kontin käynnistävä kontti näyttäytyi tarpeettomana välikappaleena prosessissa. Näkemys on subjektiivinen, ja koskee vain empiriassa rakennettua hyvin yksinkertaista toimittamisprosessia.

Koska konttiklustereita voi koostaa arkkitehtuuriltaan erilaisista suorittimista, saman ohjelmiston toimittamista yritettiin moniarkkitehtuuriseen klusteriin. Toimittaminen onnistui kehittäjän omalta laitteelta. Empiirisessä osuudessa ei kokeiltu muita automatisointipalveluita, kuten *Travisia*. Tutkimuksen aikana kävi ilmi, että

manifest-työkalun toimintoa oli tutkittu hyvin vähän. Kokeilun oli nojattava ainoastaan Dockerin tarjoamaan dokumentaatioon.

Teollisuuden ratkaisuista Balena vertautuu eniten Docker Swarmiin, sillä sen keskiössä ovat kontit. Molemmat ratkaisut tarjoavat päällisin puolin tarkasteltuna samoja toimintoja toimittamiseen ja valvontaan. Siinä missä palvelimen ja reunalaitteen yhteys Docker Swarmissa muodostetaan päällysteverkkoteknologian avulla, Balena hyödyntää VPN:ää samassa toiminnassa. Tapaustutkimus osoitti, että samassa Docker Swarm -klusterissa ohjelmistoa voi suorittaa erilaisilla suoritinarkkitehtuureilla Dockerin manifest-toimintoa hyödyntäen. Balenan ratkaisu ei tue tällaista toimintoa, vaan sovellus on käytettävissä ainoastaan projektin alustuksessa määrätyllä suoritinarkkitehtuurilla.

7.3 Mahdollisia jatkotutkimusaiheita

Tutkimuksen aikana suoritetut empiiriset kokeet koskivat vain pientä osaa operoinnin käytännöistä. Mahdollisena jatkotutkimuksen aiheena on automaattisten testien kehittäminen reunalaskennan sovellusten jatkuvan toimittamisen yhteyteen. Sovellusten toiminnan varmistaminen ennen tuotantoonvientiä on koettu vahvasti olevan osa ketterän kehityksen käytäntöjä [14]. Tietyt reunalaskennan ohjelmistot, esimerkiksi yksinkertaiset palvelut, voivat hyvinkin olla testattavissa samoin keinoin kuin palvelusuuntautuneet ohjelmistot. Fyysisten laitteiden mukanaollessa testaus saattaa kuitenkin mutkistua. Sovellus ei välttämättä käyttäydy kehityksen aikana samalla tavalla kuin reunalaitteeseen toimitettuna. Siksi sovelluksen kanssa vuorovaikutuksessa olevien esineiden toimintaa tulisi simuloida testausvaiheessa. Integraatiotestaaminen vaatisi mahdollisesti kokonaisen testausympäristön rakentamista esineineen.

Empiiristä kokeilua varten kehitetty sovellus oli hyvin yksinkertainen ja sisälsi vain yhden kovakoodatun toiminnallisuuden. Sovellus ei tehnyt lainkaan uuden tiedon vastaanottamista ja käsittelyä. Suuren kokonaisuuden kehittäminen on mahdollinen jatkotutkimuksen aihe. Esimerkiksi Docker Swarmin käytettävyyttä voisi arvioida laajamittaisesti kehittämällä MQTT-viestinvälittäjistä ja tietoa paikan päällä analysoivista ohjelmistoista maantieteellisesti hajautetun kokonaisuuden.

Kubernetesin avulla on kyetty toteuttamaan tapaustutkimus, jossa luotiin monimutkainen sovellus [41]. Tästä herää kysymys, voiko vastaavanlaisia ohjelmistoja luoda Docker Swarmia käyttäen. Tämän tutkimuksen empiirisessä osuudessa ei otettu

kantaa Kubernetesin, Docker Swarmin ja muiden konttiklustereita hallinnoivien ohjelmistojen eroihin. Mahdollinen jatkotutkimus voisikin käsitellä Kubernetesin ja Docker Swarmin eroavaisuuksia ohjelmistokehityksen ja operoinnin näkökulmasta.

Tapaustutkimuksessa esitetyistä vaatimuksista arkkitehtuuriltaan erilaisten suoritinten samanaikainen käyttö klusterissa koettiin vähiten tärkeäksi. Tällaisen klusterin rakentaminen voi kuitenkin avata uusia tutkimusaiheita. Esimerkiksi sekalaisia klustereita voisi rakentaa käytöstä poistetuista ja muista edullisesti saatavilla olevista laitteista, joiden laskentateho riittää kokonaisten käyttöjärjestelmien ja konttien suorittamiseen. Tämä aihe ei koske ainoastaan reunalaskentaa, vaan liittyy yleisesti hajautetun laskennan kokonaisuuksiin.

Lähteet

1 C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, pp. 94–100, May 2016.

2 S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *Operating Systems Review*, vol. 41, pp. 275–287, Mar. 2007.

3 R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, pp. 386–393, March 2015.

4 R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614, March 2014.

5 W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct 2016.

6 D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, pp. 49–69, May 2011.

7 M. Aoyama, "Agile software process and its experience," in *Proceedings of the 20th International Conference on Software Engineering*, pp. 3–12, April 1998.

8 L. Rising and N. S. Janoff, "The scrum software development process for small teams," *IEEE Software*, vol. 17, pp. 26–32, July 2000.

9 M. Leong, P. M. Wan, I. H. Leung, K. Leung, C. Cheung, W. M. Yeung, W. Yuen, P. W. Leong, C. Cheung, and K. K. Chow, "Cpe: A parallel library for financial engineering applications," *Computer*, vol. 32, pp. 70–77, oct 2005.

10 L. Lagerberg, T. Skude, P. Emanuelsson, K. Sandahl, and D. Ståhl, "The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 348–356, Oct 2013.

11 M. Fowler and M. Foemmel, “Continuous integration,” *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, vol. 122, p. 14, 2006.

12 L. Chen, “Continuous delivery: Huge benefits, but challenges too,” *IEEE Software*, vol. 32, pp. 50–54, Mar 2015.

13 H. H. Olsson, H. Alahyari, and J. Bosch, “Climbing the "stairway to heaven-- a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software,” in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 392–399, Sep. 2012.

14 M. Shahin, M. Ali Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

15 B. Fitzgerald and K.-J. Stol, “Continuous software engineering and beyond: Trends and challenges,” in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, RCoSE 2014, (New York, NY, USA), pp. 1–9, ACM, 2014.

16 R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, “What is devops?: A systematic mapping study on definitions and practices,” in *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP ’16 Workshops, (New York, NY, USA), pp. 12:1–12:11, ACM, 2016.

17 M. Virmani, “Understanding devops bridging the gap from continuous integration to continuous delivery,” in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pp. 78–82, May 2015.

18 H. Kang, M. Le, and S. Tao, “Container and microservice driven design for cloud infrastructure devops,” in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 202–211, April 2016.

19 A. Taivalsaari and T. Mikkonen, “A roadmap to the programmable world: Software challenges in the iot era,” *IEEE Software*, vol. 34, pp. 72–80, Jan 2017.

20 J. Ha, J. Kim, H. Park, J. Lee, H. Jo, H. Kim, and J. Jang, “A web-based service deployment method to edge devices in smart factory exploiting docker,” in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 708–710, Oct 2017.

21 M. Kasmi, F. Bahloul, and H. Tkitek, "Smart home based on internet of things and cloud computing," in *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pp. 82–86, Dec 2016.

22 J. Wei, "How wearables intersect with the cloud and the internet of things : Considerations for the developers of wearables.," *IEEE Consumer Electronics Magazine*, vol. 3, pp. 53–56, July 2014.

23 A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, pp. 22–32, Feb 2014.

24 N. Shahid and S. Aneja, "Internet of things: Vision, application areas and research challenges," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 583–587, Feb 2017.

25 "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html, 2018.

26 F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), pp. 13–16, ACM, 2012.

27 M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, pp. 854–864, Dec 2016.

28 S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, (New York, NY, USA), pp. 37–42, ACM, 2015.

29 A. Taivalsaari and T. Mikkonen, "A taxonomy of iot client architectures," *IEEE Software*, vol. 35, pp. 83–88, May 2018.

30 R. Want, B. N. Schilit, and S. Jenson, "Enabling the internet of things," *Computer*, vol. 48, pp. 28–35, jan 2015.

31 L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of things patterns," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*, EuroPlop '16, (New York, NY, USA), pp. 5:1–5:21, ACM, 2016.

32 P. Selonen and A. Taivalsaari, "Kiuas – iot cloud environment for enabling the programmable world," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 250–257, Aug 2016.

33 R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

34 R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating performance of containerized iot services for clustered devices at the network edge," *IEEE Internet of Things Journal*, vol. 4, pp. 1019–1030, Aug 2017.

35 T. Renner, M. Meldau, and A. Kliem, "Towards container-based resource management for the internet of things," in *2016 International Conference on Software Networking (ICSN)*, pp. 1–5, May 2016.

36 A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, "Exploring container virtualization in iot clouds," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 1–6, May 2016.

37 S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "Iot design patterns: Computational constructs to design, build and engineer edge applications," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 277–282, April 2016.

38 E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A framework for continuous automated iot application deployment in fog computing," in *2017 IEEE International Conference on AI Mobile Services (AIMS)*, pp. 38–45, June 2017.

39 B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*, pp. 130–135, Aug 2015.

40 M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for iot using docker and edge computing," *IEEE Communications Magazine*, vol. 56, pp. 118–123, Sep. 2018.

41 C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, "Fogernetes: Deployment and management of fog computing applications," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7, April 2018.

42 A. Buzachis, A. Galletta, L. Carnevale, A. Celesti, M. Fazio, and M. Villari, "Towards osmotic computing: Analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and iot environments," in *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, May 2018.

43 S. Hoque, M. S. d. Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 294–299, July 2017.

44 M. Großmann and C. Klug, "Monitoring container services at the network edge," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, pp. 130–133, Sep. 2017.

45 P. Tsai, H. Hong, A. Cheng, and C. Hsu, "Distributed analytics in fog computing platforms using tensorflow and kubernetes," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 145–150, Sep. 2017.

46 H. Zeng, B. Wang, W. Deng, and W. Zhang, "Measurement and evaluation for docker container networking," in *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 105–108, Oct 2017.

47 P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, (New York, NY, USA), pp. 16:1–16:10, ACM, 2017.

48 A. Ahmed and G. Pierre, "Docker container deployment in fog computing infrastructures," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 1–8, July 2018.

Liite 1. main.go

```go
package main

import (
	"encoding/json"
	"log"
	"net/http"
	"os"

	"github.com/gorilla/mux"
)

// Greeting represents the greeting
type Greeting struct {
	Message   string `json:"greeting"`
	Hostname  string `json:"hostname"`
	Name      string `json:"name"`
	TestField string `json:"testField"`
	Version   int32  `json:"version"`
}

func main() {
	log.Printf("Initializing software...")
	router := mux.NewRouter()
	router.HandleFunc("/api/greeting", GetGreeting).Methods("GET")
	log.Fatal(http.ListenAndServe(":8080", router))
}

// GetGreeting responses with a greeting in json form.
func GetGreeting(w http.ResponseWriter, r *http.Request) {
	log.Println("Received greeting request.")
	h, err := os.Hostname()
	if err != nil {
		log.Fatal("Could not get hostname!")
		w.WriteHeader(500)
	}
	greeting := Greeting{
		Message:   "Message",
		Hostname:  h,
		Name:      "Name",
		TestField: "TestField",
		Version:   1,
	}
	w.Header().Add("Content-Type", "application/json")
	w.WriteHeader(http.StatusOK)
	encoder := json.NewEncoder(w)
	encoder.Encode(greeting)
}
```

Liite 2. Dockerfile

```
1  FROM golang:latest as compiler
2  WORKDIR /go/src/github.com/[redacted]
3  ADD . /go/src/github.com/[redacted]
4  RUN go get -u github.com/golang/dep/cmd/dep
5  RUN dep ensure -v
6  RUN GOARM=7 GOARCH=arm GOOS=linux go build -o program
7
8  FROM arm32v6/alpine
9  WORKDIR /app
10 COPY --from=compiler /go/src/github.com/[redacted]program /app
11 CMD ./program
```

Liite 3. Dockerfile.x86

```
1  FROM golang:latest as compiler
2  WORKDIR /go/src/github.com/[redacted]
3  ADD . /go/src/github.com/[redacted]
4  RUN go get -u github.com/golang/dep/cmd/dep
5  RUN dep ensure -v
6  RUN GOOS=linux GOARCH=amd64 CGO_ENABLED=0 go build -o program
7
8
9  FROM alpine
10 WORKDIR /app
11 COPY --from=compiler /go/src/github.com/[redacted]program /app
12 CMD ./program
```

Liite 4. Jenkinsfile

```
def registry = 'https://registry.hub.docker.com'
def name = "[redacted]"
def dockerfile = 'Dockerfile'
def imageTag = "${name}:latest"
def newImage

pipeline {
    agent any
    stages {
        stage('Clone repository') {
            steps {
                script {
                    checkout scm
                }
            }
        }

        stage('Build image') {
            steps {
                script {
                    newImage = docker.build(imageTag, "-f ${dockerfile} .")
                }
            }
        }

        stage('Push image') {
            steps {
                script {
                    docker.withRegistry(registry, 'docker-hub-creds') {
                        newImage.push()
                    }
                }
            }
        }

        stage('Remove and prune images') {
            steps {
                script {
                    sh "docker image prune -f"
                }
            }
        }
    }
}
```

Liite 5. Docker-compose.yml

```
version: '3'
services:
  edge-api:
    build: .
    image: [redacted]:latest
    ports:
      - 8080:8080
    privileged: true
    deploy:
      mode: global
      resources:
        limits:
          memory: 50M
        reservations:
          memory: 10M
      placement:
        constraints:
          - node.labels.location == edge
```

Liite 6. Manifest-skripti

```
#!/bin/bash

docker build . -f Dockerfile -t [redacted]:arm
docker build . -f Dockerfile.x86 -t [redacted]:x86

docker push [redacted]:arm
docker push [redacted]:x86

docker manifest create [redacted]:latest \
[redacted]:arm \
[redacted]:x86

docker manifest push [redacted]:latest -p
```